......FRANCOMPUTER

LINE-BY-LINE..........

MINI MEMORY
Assembler

DISASSEMBLER

Texas Instruments

un mondo a 16 bit

INDICE

Line-by-Line Assembler

Texas Instruments

Line-by-Line Assembler

COS'E' UN'ISTRUZIONE IN ASSEMBLER

da utilizzare sia con la MINIMEMORY
e sia con l' Editor ASSEMBLER.

FORMATO DELLE ISTRUZIONI IN LINGUAGGIO SORGENTE

Un programma sorgente in Assembler è formato da istruzioni in codice che possono contenere direttive di assemblaggio,istru= zioni di macchina,pseudo istruzioni commenti.

Ogni linea (o record) di istruzioni sorgente consiste in un massimo di 80 caratteri di informazioni inclusi gli spazi. Un record può essere suddiviso in più sezioni di lunghezza varia= bile chiamati campi.
Il CAMPO ETICHETTA (label) è posizionato all'inizio dell'istru= zione sorgente e serve come punto di riferimento.

Il CAMPO OP-CODE è il codice operativo(un numero, un nome , o un'abbreviazione) della azione che deve compiere l'istruzione sorgente.

IL CAMPO OPERANDO specifica il valore su cui l'istruzione agi= sce;può essere un numero, una stringa, un indirizzo etc etc.

IL CAMPO COMMENTO è un'area in cui potete aggiungere commenti per migliorare la leggibilità del programma ma esso non influ= enza le operazioni del computer.

Le definizioni di sintassi descrivono la forma richiesta per l'uso dei comandi in relazione ai campi.

LA SEZIONE 4 descrive le procedure di scrittura e definisce i dettagli.

Nelle definizioni sintattiche relative alle istruzioni di mac= china ed alle direttive di assemblaggio vengono usate le seguen= ti convenzioni. (vedi manuale del TI pag. 31) e

° Le parentesi angolari indicano un elemento da voi de= finito.

° Il simbolo b rappresenta uno o più spazi(o blanks)

La sintassi (cioè la forma richiesta) per l'istruzione sor= gente è la seguente:

[<label>] b op-code b [< operando >] [< operando>] b[<commento>]

Come indica questa definizione di sintassi, una istruzione sor= gente può avere una etichetta da voi definita. Uno o più spazi separano l'etichetta dall' op-code. Il generico termine op-code include codici operativi mnemonici, direttive di assemblaggio, e potete dunque inserire uno di questi elementi.

Uno o più spazi separano l' op-code dall'operando, quando è ri=
chiesto un operando. Operandi addizzionali, quando servono, devo=
no essere separati da virgole. Uno o più spazi separano l'opern=
do o gli operandi dal campo commento. NOTA: Anche se la lunghez=
za massima di un record sorgente è 80 caratteri, quando lo lista=
te sono stampati solo i primi 60 caratteri di ogni linea.

CAMPO ETICHETTA

Il Campo Etichetta comincia col 1° carattere del record sorgente
e termina al primo spazio seguente. Il campo etichetta è un sim=
bolo contenente fino a sei caratteri, il primo dei quali DEVE es=
sere alfabetico. Gli altri caratteri possono essere invece alfa=
numerici. Una etichetta opzionale è per le istruzioni di macchina
e per molte direttive di assemblaggio. Quando l'etichetta è omessa
il primo carattere del record DEVE essere uno spazio.
Una istruzione SORGENTE formata solo da un CAMPO ETICHETTA è una
istruzione valida. Ha l'effetto di assegnare la locazione corren=
te all'etichetta. Di solito ciò è equivalente a piazzare l'etichet=
ta nel campo etichetta della seguente istruzione di macchina o di=
rettiva di assemblaggio.

CAMPO ISTRUZIONE

Il campo istruzione comincia dopo lo spazio che chiude il campo
etichetta oppure nel primo carattere che non è uno spazio seguente
la prima posizione del record se l'etichetta è omessa. Il campo
istruzione termina con uno o più spazi e non può superare il 60°
carattere del record sorgente.
Il campo istruzione contiene un op-code che può essere uno dei se=
guenti:

°)codice operativo mnemonico di una istruzione di macchina.
°)codice operativo di una direttiva di assemblaggio
°)simbolo assegnato ad una "extended operation" da una direttiva DXOP
°)codice operativo di una pseudo istruzione.

CAMPO OPERANDO

Il campo operando comincia dopo lo spazio che conclude il campo is=
truzione. Non può oltrepassare la 60° posizione del RECORD SORGENTE.
Il campo operando può contenere una o più espressioni,variabili o
costanti in accordo a ciò che richiede il particolare op-code. Il
campo operando termina con uno o più spazi.

CAMPO o COMMENTO e linea di COMMENTO comincia dopo lo spazio che
chiude il campo operando e può estendersi fino alla fine del re=
cord sorgente se necessario. Il campo commento può contenere qua=
lunque carattere ASCII incluso gli spazi. Unalinea o istruzione di
commento consiste in un singolo campo che comincia con un asterisco
e continua con qualunque carattere ASCII inclusi gli spazi, in qua=
lunque ordine.
Le istruzioni di commento sono stampate nel listato del codice sor=
gente ma non hanno altri effetti sull'assemblatore.Una linea com=
posta esclusivamente di spazi è considerata una linea di commento.

LINE - BY - LINE ASSEMBLER

QUELLO CHE IL MANUALE NON DICE

Il Line - by - Line Assembler é un assembler istantaneo. Ciò
significa che ogni istruzione sorgente é tradotta in codice mac-
china e memorizzata nella Mini Memory subito dopo averla scritta.
Potete immediatamente vedere il codice generato, poiché la tra-
duzione é estremamente veloce. Se fate riferimento in una istru-
zione ad una label che non avete ancora definito, l'Assembler
genera un riferimento (si vede una R). Quando più avanti la la-
bel sarà definita, l'Assembler riaggiusterà gli indirizzi davan-
ti ai vostri occhi. In ogni momento potete stampare la Tavola
dei Simboli per rivedere le label che sono state definite e quel-
le ancora sospese. Nel momento in cui terminate di scrivere il
vostro programma, il codice macchina é già nella Mini Memory
pronto ad essere eseguito. Dovete solo inserire il punto di ini-
zio del programma nella Tavola REF/DEF e riaggiustare il pun-
tatore LFAM (Last Free Andress in Memory, ultimo indirizzo libe-
ro in memoria).
L'Assembler stesso e la Tavola dei Simboli risiedono nella Mini
Memory. Ciò aumenta la sua velocità, ma riduce lo spazio dispo-
nibile per il vostro programma.
Il Line - by - Line Assembler usa praticamente le locazioni com-
prese tra >7II8 (28952) e >7CD7 (3I959). La tavola dei Simboli
parte dalla locazione >7CD8 (3I960) e si espande verso >7FFF
(32767). (La sua lunghezza dipende dal numero di label che usa-
te nel programma). Il punto di inizio di default per il vostro
codice oggetto é >7D00 (32000), il che lascia spazio nella Ta-
vola dei simboli per nove labels. Inoltre per usare il Line -
by - Line Assembler dovete avere i suoi due punti di entrata
(OLD e NEW) nella Tavola REF/DEF, insieme a quello del program-
ma che state scrivendo. Ciò significa che gli indirizzi compre-
si tra >7FE8 (32744) e >7FFF (32767) sono occupati dalla Tavola
REF/DEF. Quindi con le precedenti suddivisioni avete a disposi-
zione 744 bytes per il vostro programma (da >7D00 a >7FE7).
Se non usate nessun nome simbolico, il vostro programma può par-
tire dalla locazione >7CE0 (3I968) guadagnando così 32 bytes,
ma é molto difficile programmare senza labels o simboli. Se in-
vece volete usare più di nove labels, dovrete far partire il
vostro programma da indirizzi più grandi di >7D00 (con una di-
rettiva AORG), ma ciò chiaramente fa diminuire lo spazio dispo-
nibile. Il semplice programma LINES compreso nella confezione
della MINI MEMORY illustra un punto importante riguardo la lun-
ghezza del programma. Quando caricate per la prima volta i pro-
grammi da nastro, sia il Line by Line Assembler che Lines sono
presenti nella Mini Memory. Le istruzioni di programma e le aree
dati di Lines occupano le locazioni comprese tra >7D00 e >7FE7.
Se eseguite il programma Lines l'Assembler non é più presente in
memoria, perché Lines usa come area di lavoro delle locazioni
comprese tra >7II8 e >7D00. Questo fatto mette in chiaro due
punti: I. non é necessario che l'Assembler sia presente in memo-
ria durante l'esecuzione di un programma scritto da voi e 2. po-
tete usare altre aree di RAM come Workspace quando eseguite il
programma se non lo inizializzate in anticipo. In questo modo
potete usare l'Assembler per scrivere programmi che sono in realtà
molto più ampi di 744 bytes.

LINE - BY - LINE ASSEMBLER

INTRODUZIONE

La cassetta audio inclusa nella confezione della Mini Memory contiene un Assemblatore simbolico linea - a - linea ed un programma dimostrativo grafico chiamato Lines. L'assemblatore linea - a - linea vi permette di scrivere istruzioni in linguaggio sorgente Assembly del TMS9900, unalinea alla volta direttamente dalla tastiera della vostra console. Il programma dimostrativo Lines traccia automaticamente linee colorate sullo schermo del computer.

Quando caricate un programma Assembler nel Modulo Mini Memory, ogni istruzione sorgente che scrivete viene immediatamente assemblata (tradotta) nel codice oggetto ed inserita nelle locazioni di memoria da voi specificate. Quindi una volta completata la scrittura del vostro programma e scritto il suo nome e l'indirizzo di partenza nella tavola REF/DEF, esso é pronto per essere eseguito.

NOTA IMPORTANTE: Siccome ogni volta che scrivete una linea di programma le istruzioni vengono assemblate e memorizzate in modo diretto, siate sicuri che le locazioni di memoria specificate nel vostro programma siano disponibili. In caso contrario nessun codice sarà generato o memorizzato.

Anche se l'Assembler converte ogni istruzione in codice macchina una linea alla volta nel momento in cui viene inserita, i codici sorgente sono memorizzati in un Buffer di testo lungo nove pagine. Avete quindi la possibilità di rivedere le linee di programma già scritte muovendo lo schermo in verticale con i tasti "Freccia in su" e "Freccia in giù".

Questo manuale illustra le possibilità dell'Assembler linea-a-linea assumendo che il lettore già conosca la programmazione in Assembler. Per una completa guida all'Assembler del TMS9900 vi consigliamo di consultare il Manuale dell'Editor/Assembler. Inoltre in questo manuale sono descritte le modalità per caricare ed eseguire il programma dimostrativo Lines. (Vedi la sezione "Caricare l'Assembler linea - a - linea")

Carichiamo l'Assembler Line - by - Line

Sia il Line - by - Line Assembler che il programma grafico LINES vengono caricati da nastro nello stesso tempo attraverso il comando L (LOAD) dell'opzione EASY BUG Debugger. I passi che seguono descrivono le operazioni di caricamento.

I. Con il modulo Mini Memory inserito nella console, collegate il vostro registratore a cassette nel modo descritto dal Manuale d'uso del Computer.

2. Premete un tasto qualunque per far apparire la lista di selezione principale e scegliete l'opzione MINI MEMORY. Quando appare la lista di selezione della MINI MEMORY, premete 3 (REINITIALIZE) per preparare la memoria al caricamento di un nuovo programma. Quindi premete QUIT per tornare al Titolo Principale.

3. Inserite la cassetta dell'Assembler nel registratore e riavvolgete il nastro.

4. Premete un tasto qualunque per far apparire la lista di selezione principale e selezionate l'opzione EASY BUG.

5. Quando appare la descrizione dei comandi dell'EASY BUG, premete un tasto qualunque per cancellare lo schermo. Quindi scrivete L e premete ENTER per far partire la procedura di caricamento. Da questo momento in poi il computer stampa delle istruzioni per aiutarvi a procedere.

6. Dopo che il programma Assembler é stato caricato, premete QUIT per tornare al Titolo Principale. Quindi premete un tasto qualsiasi per far apparire la lista di selezione principale, e selezionate l'opzione MINI MEMORY.

7. Quando appare la lista di selezione di MINI MEMORY, premete 2 per scegliere l'opzione RUN. Lo schermo si cancella ed il computer vi chiede il nome del programma che volete eseguire.

   A. Se volete eseguire il programma LINES, scrivete LINES e premete ENTER. Questo programma traccia linee colorate sullo schermo. Se premete il tasto C, potete "congelare" il colore della linea appena tracciata, e tutte le linee seguenti avranno lo stesso colore. Premendo ancora C cancellate questa possibilità. Per fermare il programma, premere QUIT.

B. Se volete scrivere un nuovo programma in linguaggio Assembly scrivete NEW e premete ENTER. Il computer entra in ambiente Assembler, cancella la Tavola dei Simboli (di cui parleremo più avanti) e aspetta che scriviate la vostra prima linea di programma.

C. Se volete continuare a scrivere o modificare un programma esistente, scrivete OLD e premete ENTER. La Tavola dei Simboli preesistente sarà conservata ed il computer mostrerà la prima locazione di memoria libera pronto a continuare il vostro programma.

Nota: nel vostro modulo MINI MEMORY possono essere già caricati i programmi LINES e Assembler. Per accertarvene, selezionate l'opzione RUN della MINI MEMORY e scrivete il nome di programma appropriato quando appare la richiesta del computer. Se é presente il programma sarà eseguito immediatamente. Se il programma invece non é stato ancora caricato, il computer stampa la frase "PROGRAM NOT FOUND".

Potete anche vedere il codice del programma LINES utilizzando il comando M dell'EASY BUG. Scrvendo solo M7CD6 e premendo ENTER, potete stampare tutte le linee di codice senza modificarle.

NOTA IMPORTANTE: Quando scrivete ed assemblate un programma, la tavola dei simboli si può sovrapporre a parti del programma LINES. Quando vorrete eseguire ancora LINES bastera semplicemente ricaricare il programma da nastro nella memoria del modulo.

LINE - BY - LINE ASSEMBLER

Sintassi dell'Assembler

Ogni linea (o record) del vostro programma sorgente é composta da quattro sezioni chiamate CAMPI. Questi campi, se presenti (alcuni sono opzionali), devono essere scritti nell'ordine e nel formato (sintassi) richiesti dal programma Assembler. In questo manuale nelle definizioni di sintassi relative alle istruzioni ed alle direttive di assemblaggio si applicano le seguenti convenzioni:

I. I campi in lettere maiuscole, inclusi i caratteri speciali devono essere scritti esattamente come appaiono.

2. I campi in parentesi quadre sono opzionali.

3. I campi in parentesi angolari sono obbligatori.

4. Una b minuscola indica uno spazio.

5. Una b minuscola seguita da tre punti (b...) indica uno o più spazi.

La sintassi generale di una istruzione Assembler é la seguente:

[label] b...<opcode>b [operando] [,operando] [b...commento]

Il campo label (etichetta) deve contenere uno o due caratteri di cui il primo alfabetico oppure se omettiamo una label deve contenere uno spazio. Il secondo carattere di una label (se presente) può essere alfanumerico. La label é seguita da uno o più spazi. Se non scrivete l'etichetta, una pressione della Barra Spaziatrice sposta il cursore automaticamente all'inizio del campo opcode.

Il campo opcode contiene il Codice Operativo dell'azione che deve essere eseguita attraverso l'istruzione sorgente. Questo campo é composto da uno fino a quattro caratteri alfabetici, come ad esempio A per Add oppure AORG per la direttiva Absolute origin. Esso é seguito da un solo spazio.

Il Campo Operando contiene uno o due operandi a seconda del tipo di istruzione specificata nel campo Opcode. Notate che il campo operando non richiede spazi all'interno di esso, ed operandi multipli sono separati solo da virgole. Il campo operando si chiude premendo la Barra Spaziatrice (ed il cursore si posiziona sul campo commento) oppure premendo ENTER (così si chiude la linea). Se una istruzione non richiede operandi, il campo operando si omette.

Il Campo Commento può includere qualsiasi carattere, e continua finché non premete ENTER per terminare la linea di programma.

LINE - BY - LINE ASSEMBLER

Il Line - by - Line Assembler predefinisce alcuni simboli. Quando un operando include il simbolo del Dollaro ($) come carattere iniziale, esso é considerato riferirsi al contatore di locazione. Per esempio, alla locazione>7D00, l'istruzione

```
          JMP  $+8
```

e l'istruzione

```
          JMP  >7D08
```

sono considerate equivalenti. Quando specificate registri come operandi, potete usare il simbolo R seguito da un numero decimale. Tuttavia le istruzioni

```
          MOV  R2,RI5
```

e

```
          MOV  2,I5
```

sono equivalenti.

Nota : il sistema di numerazione standard per il Line - by - Line Assembler é il decimale; i numeri esadecimali sono indicati del prefisso "maggiore di" (>).

DIRETTIVE DI ASSEMBLAGGIO

Questa sezione descrive le sette direttive di assemblaggio riconosciute dall' assembler Line - by - Line. Una direttiva non deve essere confusa con una istruzione del linguaggio Assembly, che dice al Microprocessore di eseguire solo una singola istruzione, come ad esempio Add o Move. Le direttive sono invece comandi di aiuto alla programmazione che dirigono il programma Assembler ad eseguire certe istruzioni, e l'Assembler può eseguire più istruzioni per soddisfare una sola direttiva. (Per una descrizione delle istruzioni del linguaggio Assembly del TMS 9900 consultare il manuale dell' EDITOR / ASSEMBLER).
Le direttive che descriveremo sono :

AORG	Absolute Origin
BSS	Block Starting with Symbol
DATA	Word Initialization
END	End Program
EQU	Assembly - Time Constant Definition
SYM	Symbol Table Dysplay
TEXT	String Constant Initialization

AORG - Absolute Origin

Formato : [label] AORG<indirizzo>

Questa direttiva può essere usata per posizionare il contatore di locazione ad un valore specifico (che deve essere sempre un indirizzo pari) durante le operazioni in assembler. Generalmente viene usata come prima azione di un programma per selezionare la locazione di partenza del codice assemblato; comunque può essere usata in qualunque momento durante la scrittura di un programma.

Esempio :

AORG	>7D80	Ottiene che la successiva istruzione assemblata sia memorizzata partendo dalla locazione >7D80.

BSS - Block Starting with Simbol

Formato : [label] BSS <numero di bytes da riservare>

La direttiva BSS riserva un blocco di memoria (per la memorizzazione di variabili o per area di lavoro dei registri) senza inizializzarlo. Partendo dall'indirizzo specificato nella label, l'Assembler incrementa il contatore di locazione del numero di bytes specificato nella direttiva.

Il numero di bytes deve essere zero o positivo. Il valore risultante nel contatore di locazione é arrotondato per eccesso ad un numero pari se necessario. In altre parole il bit meno significativo dell'indirizzo viene troncato se il valore risultante é dispari.

Esempio:

WS	BSS 32	Supposto che WS si riferisca alla locazione 7D00, viene incrementato il contatore di locazione fino a 7D20 riservando un blocco di 32 bytes come area di lavoro.

DATA : Word Initialization

Formato : [label] DATA <valore>

La direttiva DATA vi permette di inizializzare una o più parole di memoria ad un particolare valore. Questa direttiva é particolarmente utile quando dovete inserire una serie di dati come parte del vostro programma. Potete inserire una direttiva DATA, nel campo OPCODE seguita da una costante o da un simbolo come operando, in un punto qualunque del vostro programma.

Gli operandi, per una direttiva DATA possono essere: un riferimento irrisolto (unresolved reference), una costante numerica, un simbolo definito, una sequenza di costanti numeriche e di simboli definiti uniti dal simbolo "più" (+) o dal simbolo "meno" (-). Nell'ultimo caso (una sequenza di somme e sottrazioni) non viene considerato il riporto ne segnalato un eventuale overflow.

Esempi:

DATA >I234	Inizializza la locazione al valore >I234.
DATA AX	Se AX= >3456, inizializza la locazione a >3456.
DATA GH	Se GH é un riferimento irrisolto seguente, la locazione sarà inizializzata al valore corrispondente a GH quando GH sarà definito.
DATA 2+5-3	Inizializza la locazione al valore 4. (E' equivalente a DATA 4).

La direttiva DATA può ammettere come operandi anche una sequenza di costanti separate da virgole.

DATA [costante (definita o indefinita), costante,...,]

Notate che una costante non definita é accettabile SOLO se é il primo operando della lista.

Le direttive BSS e DATA hanno funzioni similari. Tuttavia la direttiva BSS riserva semplicemente spazio in memoria senza inizializzarlo, mentre la direttiva DATA riserva spazio in memoria inizializzandolo ad un valore (o valori) specificato.

END : End Program

Formato : END

L'Assembler può essere lasciato in ogni momento scrivendo nel campo Opcode la direttiva END. Quando essa viene inserita, l'Assembler mostra il numero di riferimenti irrisolti (se ce ne sono). Se esistono riferimenti irrisolti, tornate all'Assembler e risolveteli prima di lasciare il programma. Se non lo fate, nel vostro programma potranno risultare codici operativi non validi. La direttiva SYM (descritta più avanti) può aiutarvi ad identificare i riferimenti irrisolti prima che terminiate di scrivere il vostro programma.

Dopo aver risolto tutti i riferimenti, l'Assembler mostra la frase :

OOOO UNRESOLVED REFERENCES

A questo punto premendo ENTER uscite dall'Assembler e tornate alla lista di selezione della Mini Memory. Premendo un qualunque tasto meno ENTER, si ottiene che l'Assembler attenda una vostra successiva istruzione.

EQU : Assembly Time Constant Definition

Formato : <label> EQU <costante definita>

La direttiva EQU serve ad assegnare il valore di un simbolo definito ad un altro simbolo e per definire il valore di una costante simbolica.

Nota : non é prevista alcuna direttiva per cambiare il valore di un simbolo dopo averlo definito.

Esempi :

CD	EQU >A55A	Assegna a CD il valore >A55A
FG	EQU I5	Assegna a FG il valore decimale I5
A	EQU FG	Pone A uguale a FG

SYM : Symbol Table Display

Formato : SYM

La direttiva SYM vi permette di rivedere, in qualunque momento durante la scrittura del vostro programma, i simboli di riferimento ed i valori a loro associati (se ce ne sono) che avete fino ad allora usato nel programma.

La tavola dei Simboli viene stampata in tre categorie :

RESOLVED REFERENCES (Riferimenti risolti). Ogni label o variabile già definita (a cui é già stato assegnato un valore).

UNRESOLVED REFERENCES (WORD) Ogni label o variabile già introdotta in istruzioni di programma che non sono salti ma non ancora definita.

UNRESOLVED REFERENCES (JUMP) Ogni label che é stata introdotta in istruzioni di salto e non ancora definita.

Se una delle categorie precedenti non ha simboli ad essa associati, non viene stampata. Se un simbolo risulta non risolto (é stato introdotto ma non ancora definito) viene stampato anche l'indirizzo dell'istruzione in cui compare. Quando la Tavola dei simboli si riempie, la direttiva SYM viene cancellata e l'Assembler attende la vostra successiva istruzione.

Esempio :

Locazione		Istruzione	Commenti
		AORG >7D00	Scelta dell'indirizzo di partenza
7D00 0000	WS	BSS 32	Riserva area di lavoro
7D20 020I		LWPI WS	Carica l'area di lavoro
7D22 7D00			
7D24 020I		LI R2,W2	Assegna ad R2 un valore indefinito
7D26R0000			
7D28RIOFF		JMP J2	Salto ad un indirizzo indefinito
7D2A XXXX		SYM	Stampa la tavola dei Simboli

RESOLVED REFERENCES
WS - 7D00

UNRESOLVED REFERENCES (WORD)
W2 - 7D26

UNRESOLVED REFERENCES (JUMP)
J2 - 7D28

7D2A XXXX (XXXX é il dato esistente in memoria). L'Assembler attende la vostra istruzione seguente.

Se un simbolo non definito viene introdotto in più di una istruzione, viene stampato il simbolo e l'indirizzo di ciascuna istruzione fino ad un massimo di 32.

TEXT : String Constant Initialization

Formato : [label] TEXT '<stringa di caratteri>'

La direttiva TEXT vi permette di inserire una stringa di caratteri, di tradurla in codice ASCII e di memorizzarla come parte del vostro programma. Ogni carattere stampabile, eccetto l'apostrofo, può essere inserito come parte di una istruzione TEXT, ed il codice ASCII che viene memorizzato é esattamente quello del carattere che inserite. Notate che i tasti di controllo e di funzione speciali (AID, REDO, etc.) generano codici ASCII validi (compresi tra>0 e>F) che vengono memorizzati ma non stampati sullo schermo.

La stringa relativa ad una direttiva TEXT può essere lunga a piacere, e deve essere preceduta e seguita da un apostrofo. Se viene inserito un numero dispari di caratteri ASCII, alla stringa viene aggiunto un byte nullo (>00) per spostare il contatore di locazione al valore pari seguente.

Esempio :

TEXT 'ABCD'	Memorizza i valori> 4142 e>4344 nelle corrispondenti locazioni di memoria.

Nota : la funzione ERASE non cancella dalla memoria eventuali caratteri già scritti.

LA TAVOLA DEI SIMBOLI

L'Assembler vi permette di usare sia in istruzioni che in salti etichette e variabili non ancora definite o risolte a patto che esse vengano definite o risolte più avanti nel corso del programma. L'Assembler prende nota di tutti i simboli definiti o introdotti in un programma e memorizza queste informazioni in una Tavola Dei Simboli.

La Tavola dei Simboli é divisa in tre parti : riferimenti a simboli definiti (defined symbol reference), riferimenti ad istruzioni non risolte (unresolved word references), riferimenti a salti non risolti (unresolved jump references). Anche il numero di elementi della tavola dei simboli viene memorizzato. Siccome ogni riferimento é memorizzato in 4 bytes (i primi due si chiamano Label Word, gli ultimi due Andress Word), la lunghezza fisica della Tavola é quattro volte il numero dei riferimenti.

La Tavola dei Simboli comincia alla locazione di memoria>7CD8. Siccome ogni elemento della Tavola dei Simboli occupa quattro bytes, assicuratevi che l'indirizzo di inizio del vostro programma riservi spazio adeguato al numero di elementi della Tavola, richiesti dal vostro programma. In caso contrario, quando il vostro programma sarà eseguito la Tavola dei Simboli potrà sovrapporsi alla parte iniziale del vostro codice oggetto.

Riferimenti a Simboli Definiti

Se in una istruzione compare una label risolta o un altro simbolo definito, la Label Word memorizzata nella Tavola dei Simboli é semplicemente il codice ASCII corrispondente al simbolo. Se il simbolo é formato da un solo carattere, esso viene memorizzato aggiungendo il codice ASCII del simbolo uno. Per esempio se il simbolo é A, esso sarà memorizzato come >3I4I. La seconda parola (Andress Word) contiene il valore esadecimale corrispondente alla costante definita oppure la locazione di memoria a cui la label si riferisce. Per esempio se la costante AC viene definita essere >8375, i valori memorizzati saranno :

4I43	Nome del simbolo definito
8375	Valore corrispondente

Riferimenti ad istruzioni non risolte

I valori memorizzati nella Tavola dei Simboli per un riferimento ad una istruzione non risolta sono simili a quanto detto prima, solo che il bit più significativo della Label Word viene attivato e la Andress Word punta all'ultima locazione nella quale il simbolo non definito é stato usato. Per esempio, se la label AC viene usata come riferimento non risolto nella locazione >7E00, e nessun riferimento successivo a questa label esiste nel vostro programma, i valori memorizzati nella Tavola dei Simboli sono :

CI43	Nome del simbolo non definito
7E00	Indirizzo

Riferimenti a salti non risolti

I valori memorizzati per un riferimento ad un salto non risolto sono simili ai precedenti, solo che viene attivato il bit più significativo DEL BYTE MENO SIGNIFICATIVO della Label Word e l'Andress Word punta alla locazione dell'ultima istruzione di salto che usa quella label non risolta. Per esempio se nella locazione >7D00 esiste un riferimento di salto non risolto alla label AC e non sono stati inseriti altri riferimenti a questa label, i valori memorizzati nella Tavola dei Simboli sono :

4IC3	Nome della label non definita
7D00	Indirizzo

Il byte meno significativo di una istruzione di salto non risolta indica la distanza (in numero di parole) dal più recente riferimento precedente di salto non risolto alla stessa label. Se non ci sono riferimenti precedenti, al byte é assegnato il valore -1 (>FF).

Numero massimo di Riferimenti non Risolti Stampati

La prima volta che l'Assembler memorizza un simbolo non risolto nella Tavola dei Simboli, i caratteri che formano il simbolo sono inseriti nella Tavola seguiti dall'indirizzo nel quale il simbolo é stato introdotto. Questo indirizzo é chiamato puntatore. Il contenuto del puntatore é posto a zero, indicando così che quello é il primo riferimento al simbolo non risolto.

Per esempio consideriamo il programma seguente quando viene assemblato :

```
7D00 02E0         LWPI WS
7D02R0000
7D04 C820         MOV @WS,@DG
7D06R7D02
7D08R0000
                  SYM
UNRESOLVED REFERENCES (WORD)
WS-7D06 WS-7D02 DG-7D08
```

In questo esempio, il contenuto del primo riferimento a WS (alla locazione>7D02) é posto a zero per indicare che quello é il primo riferimento a quel simbolo. Il contenuto del seguente riferimento non risolto a WS (alla locazione>7D06) ha il valore di>7D02 (l'indirizzo del precedente riferimento a WS).

Come risultato di un errore di battuta durante la scrittura di un programma, un riferimento non risolto può sembrare avere un numero indefinito di puntatori. Il seguente segmento di programma può fare da esempio :

```
7D00          W1  EQU >1234
7D00 020I         L1  R1,WS
7D02R0000
7D04              AORG>7D00
7D00 020I         L1  R1,W1
7D02 I234
```

Nell'esempio precedente, WS appare nella Tavola dei Simbòli come un riferimento ad una istruzione non risolta con un puntatore a >7D02. La seguente direttiva AORG assegna alla locazione >7D02 il valore >I234. Di conseguenza ora esiste un numero indefinito di puntatori a WS poiché l'assembler considera il valore >I234 della locazione >7D02 come puntatore ad un riferimento precedente e così via.

Per prevenire, in casi come il precedente, la possibilità di stampa di un numero indefinito di riferimenti, la direttiva SYM stampa un massimo di 32 riferimenti per ogni simbolo non risolto.

TECNICHE DI CORREZIONE

Come già detto, l'Assembler conserva i codici sorgente in un buffer di nove pagine, per permettervi di rivedere le linee di programma già scritte. Quando ragguingete la fine del buffer, il titolo del Line - by - Line Assembler appare in basso sullo schermo per avvertirvi che il buffer é pieno ed é tornato al punto di partenza. Ogni nuovo codice sorgente che scriverete, ora cancellerà un codice sorgente precedentemente scritto. D'altra parte vi consigliamo a questo punto di rivedere il vostro codice sorgente usando i tasti "freccia in su" e "freccia in giù" per muovere le linee sullo schermo.

Se trovate un errore nel vostro codice sorgente prendete nota dell'indirizzo della linea contenente l'errore. Quindi usate la direttiva AORG per tornare all'indirizzo annotato e riscrivete la linea corretta.

Potete inoltre correggere errori di battuta mentre state scrivendo una linea premendo ERASE o usando uno dei metodi che descriveremo adesso.

Una label, sia nel campo etichetta che in quello operando, può essere corretta semplicemente continuando a scrivere i simboli esatti prima di premere la Barra Spaziatrice per uscire dal campo. Per esempio, se avete scritto VF al posto di CD come label, premete semplicemente CD prima di premere la Barra Spaziatrice per passare al campo successivo. L'Assembler accetta gli ultimi due caratteri inseriti nel campo come label corretta. Se volete correggere una label di un solo carattere, scrivete I (uno) per undicare il numero di caratteri usati, quindi scrivete il carattere alfabetico esatto prima di premere la Barra Spaziatrice.

Un metodo simile può essere usato per correggere un valore esadecimale scritto nel campo operando. Per esempio se scrivete >I234 al posto di> 2234, semplicemente continuate a scrivere il valore corretto. In altre parole quello che dovrete scrivere sarà >I2342234. L'Assembler considera gli ultimi quattro caratteri come valore esadecimale corretto. La correzione di valori decimali é invece spesso molto difficile, poiché l'Assembler considera gli ultimi sedici caratteri scritti come valore esatto. Se il vostro numero decimale é più corto di sedici cifre, dovrete scrivere tanti zeri quanti ne occorrono per raggiungere sedici cifre e poi battere il numero esatto. Risulta chiaro che in questi casi é probabilmente meglio cancellare l'ultima linea con ERASE e quindi riscriverla correttamente.

Quello che scrivete nel campo opcode si può correggere solo premendo ERASE per cancellare l'intera linea e quindi riscrivendo la linea corretta.

CONDIZIONI DI ERRORE

Durante la scrittura di un programma esistono tre condizioni che provocano la comparsa di un messaggio di errore sullo schermo.

Condizione	Messaggio Stampato
I. Cercate di ridefinire una label già definita.	°ERROR°
2. Scrivete un codice operativo o una direttiva inesistente.	°ERROR°
3. Andate oltre lo spazio consentito con una istruzione di salto.	°R-ERROR°

Ogni messaggio di errore é accompagnato dall'emissione di un suono ed é stampato sulla stessa linea dell'istruzione che ha causato l'errore. Premete un tasto qualunque per cancellare la linea. (Il contatore di locazione non sarà modificato).

Se inserite una istruzione di salto ad una label non definita e più avanti vi accorgete che la definizione della label provoca che la precedente istruzione di salto sia fuori dallo spazio consentito, l'indirizzo stampato a sinistra del messaggio °R-ERROR° é l'indirizzo della istruzione di salto. Continuate a premere ENTER, per vedere gli altri (eventuali) indirizzi di istruzioni di salto fuori spazio consentito dello stesso livello.

Se prevedete che una istruzione di salto ad una label non definita possa risultare fuori dallo spazio consentito, é una buona idea far seguire alla istruzione di salto due istruzioni NOP (no-operation), per permettervi di modificare il vostro programma nel caso ci fosse l'errore. Il seguente segmento di programma illustra questa procedura.

```
7DOO XXXX          JNE J2
7DOORI6FF
7D02 IOOO          NOP
7D04 IOOO          NOP
7D06 CO8I          MOV R1,R2
  .
  .
  .
7EIO XXXX      J2  EQU $
7DOO °R-ERROR°
```

(Premete ENTER per stampare altri eventuali indirizzi di istruzioni di salto errate; quindi premete un tasto qualunque per uscire dalla condizione di errore.)

```
7EIO XXXX          AORG>7DOO
7DOO I302          JEQ $+6
```

Notate che la logica di salto del programma corretto é opposta all'originale.

```
7D02 0460          B @J2
7D04 7EIO
```

Eseguite il vostro programma.

Dopo aver assemblato il programma, il suo nome ed indirizzo di partenza devono essere aggiunti alla tavola REF/DEF in modo che la MINI MEMORY possa trovare il programma ed eseguirlo. Un modo di inserire il nome e l'indirizzo di partenza é usare il sottoprogramma LOAD dal TI BASIC. (Consultare il paragrafo "Additional TI BASIC Subprograms" del manuale della MINI MEmory).

Un altro modo é quello di usare le Direttive di Assemblaggio. Per prima cosa dovete stabilire se é disponibile lo spazio necessario ad aggiungere il nome del vostro programma alla Tavola REF/DEF. Dopo aver scritto l'ultima linea del vostro programma, usate la direttiva AORG per leggere dalla memoria il Primo Indirizzo Libero (FFAM) e l'Ultimo Indirizzo Libero (LFAM) del Modulo. Le locazioni che contengono queste due variabili sono >7OIC e >7OIE rispettivamente. Sottraete il valore contenuto in >7OIC dal valore contenuto in >7OIE; se la differenza é maggiore di 7 Bytes, avete ancora posto per memorizzare il nome del vostro programma.

Dopo esservi assicurati che é disponibile spazio sufficiente, per aggiungere il nome del vostro programma alla REF/DEF Table, sottraete 8 dal vecchio LFAM ed inserite (poke) il nuovo valore di LFAM nella locazione >7OIE usando la direttiva DATA. In questo modo avete riservato lo spazio che vi serve nella tavola REF/DEF.

Il nome del programma può essere lungo da uno a sei caratteri, però il nome del programma da inserire nella tavola REF/DEF DEVE essere lungo ESATTAMENTE sei caratteri. Se il nome del vostro programma é lungo meno di sei caratteri, dovrete aggiungere degli spazi quando lo scrivete nella tavola.

Usate la direttiva AORG per portarvi sul nuovo punto di inizio della Tavola ed inserite il nome del programma attraverso una direttiva TEXT. Dopo aver scritto il nome del programma, il contatore di indirizzo avanza fino alla seguente parola disponibile dove deve essere memorizzata la locazione di partenza del programma (due Bytes). Usate una direttiva DATA per inserire questa ultima informazione.

ESEMPIO : Determiniamo lo spazio di memoria rimanente.

L'esempio seguente presuppone che abbiate appena scritto il "Semplice Programma" compreso nel manuale della MINI MEMORY e non siate usciti dall'Assembler.

Lo schermo mostra	Voi scrivete	Commenti
7F04 XXXX	AORG>7IOC	>7F04 rappresenta il corrente indirizzo del contatore di locazione e XXXX un dato qualunque che l'indirizzo contiene.
70IC XXXX		XXXX rappresenta l'indirizzo del vecchio FFAM.
70IC 7F04	DATA>7F04	L'istruzione DATA inserisce il nuovo FFAM che é il primo indirizzo libero immediatamente seguente il programma.
70IE 7FE8		>7FE8 rappresenta l'indirizzo attuale del LFAM. Sottraete FFAM da questo valore. Se il risultato é 7 bytes o più grande, avete ancora spazio per il nome del programma.
70IE 7FE0	DATA>7FE0	Sottraete 8 bytes dal vecchio LFAM e memorizzate il risultato come nuovo LFAM attraverso una direttiva DATA (>7FE0 rappresenta il nuovo LFAM.)
7020 XXXX		Il contatore avanza alla locazione seguente e stampa un qualsiasi dato assemblato.

ESEMPIO : Inserite il nome del programma e l'indirizzo di partenza.

L'esempio seguente che comincia dove finisce quello precedente, vi mostra come inserire il nome e l'indirizzo del programma DISP$ (vedi "Semplice Programma" nel manuale della MINI MEMORY) alla tavola REF/DEF.

Lo schermo mostra	Voi scrivete	Commenti
7020 XXXX	AORG >7FE0	Vi porta sul nuovo punto di inizio della Tavola.
7FE0 XXXX		XXXX rappresenta un qualsiasi dato attualmente memorizzato alla locazione >7FE0.
7FE0 4449 7FE2 5350 7FE4 2420	TEXT 'DISP$'	Inserisce il nome del programma: DISP$. (Notate che il carattere spazio viene aggiunto per arrotondare la lunghezza del nome a 6 caratteri). I caratteri del nome, compreso lo spazio, vengono memorizzati in 6 bytes a partire dalla locazione >7FE0.
7FE6 XXXX		Il contatore avanza alla locazione successiva, dove sarà inserito il punto di entrata del programma, e mostra il valore attualmente contenuto dalla locazione.
7FE6 7E20	DATA DS	La label DS (uguale a 7E20) indica il punto di inizio del programma.

Adesso siete pronti per uscire dall'Assembler e far girare il vostro programma.

MEMORIZZATE IL VOSTRO PROGRAMMA SU NASTRO

Per memorizzare un programma su nastro per prima cosa premete QUIT per lasciare l'opzione MINI MEMORY, quindi selezionate l'EASY BUG ed usate il comando S. Per ottenere i migliori risultati scrivete l'indirizzo di partenza >7000 e l'indirizzo di arrivo >7FFF per essere sicuri che la Tavola REF/DEF ed i puntatori alla Low Memory siano memorizzati. In caso contrario si renderà necessario inserire il nome del vostro programma nella Tavola REF/DEF ogni volta che lo caricate.

Per dettagli addizionali, consultate la sezione "EASY BUG Debugger" del manuale della MINI MEMORY.

Plotting With The Home Computer

By Joseph G. DeVincentis, Jr.
P. O. Box 375
University of Dallas
Irving, TX 75061

The Assembly Language routines presented in this article will let Home Computer users draw axes, plot curves, and even draw objects in perspective. The software, consisting of plotting routines for the Mini Memory Cartridge, accesses the powerful graphics capabilities of the TI-99/4A through TI BASIC.

The routines supplied in this package require either of the following peripheral configurations:

1. Memory Expansion, cassette recorder, Mini-Memory, and assembled (object file) routines available on this issue's "99'er Magazine-On-Tape." [See page 69]
2. Memory Expansion, disk system, Mini-Memory, Editor/Assembler, and source files (hand entered from listings included with this article.)

Note: The large size of these subroutines, and the fact that part of Mini-Memory is required for the Line-by-Line Assembler, dictates that the *source* files be assembled via the Editor/Assembler rather than the Line-by-Line Assembler that comes with the Mini-Memory. We are therefore making the *object* file available on tape so that readers without an Editor/Assembler and disk system can take advantage of this powerful software—Ed.

Theory of Operation

The routines work on the basis of a plotter. The screen is the plotter's surface, and the routines control an imaginary "pen." The pen can be moved with or without drawing a line on the screen, and if it tries to draw off the screen, it will change no data other than the position of the pen.

The graphic capabilities are unique to the Video Display Processor (VDP) in the 99/4A. However, if the VDP in a 99/4 is changed from a TMS9918 to a TMS9918A (the chip in a 99/4A), these routines are also usable with the 99/4. The TMS9918 and TMS9918A are pin compatible and any software designed for the 99/4 will run with the new chip in place, with no modifications. [This chip swapping should only be done by a competent technician—Ed.]

The resolution of the screen is 192 pixels (dots) vertically and 256 horizontally. The routines allow you to clear the screen, *scale* the screen, draw X and Y axes, output text, change the pen's position and draw lines. Because the routines take advantage of the advanced *graphics mode* of the TMS9918A Video Display Processor (VDP), compromises had to be made (due to memory requirements in this mode).

The VDP takes up 12K bytes of space in VDP RAM to define the shape and color of the characters. This leaves very little room for BASIC programs. Therefore, a buffer was created in the Memory Expansion to keep the character shape table. Because this buffer is *not* in the VDP RAM, the plot cannot be seen until the GRAPH command is issued. Also, once this command is issued, the BASIC program will be destroyed. Therefore, it is advisable to *save your BASIC programs before running them!*

As a positive side effect, after the plot is put on the screen and the user returns to the power-up screen (by pressing the Q key), the plot will remain unchanged in the Memory Expansion Buffer. The plot will exist as long as the user does not issue the GCLEAR command or turn off power to the console or the Memory Expansion. Therefore, it is possible to add data to a plot even after looking at it several times.

The line-drawing algorithm is based on Bresenham's algorithm described in *Principles of Interactive Graphics* by William M. Newman and Robert F. Sproull (McGraw-Hill Book Company). The algorithm was originally designed for control of a digital plotter but is easily adapted to the type of display used on the 99/4A. For a complete discussion of this algorithm, refer to pages 25 and 26 of the book mentioned.

Within the line-drawing routine of the package, A and DELTAA refer to the axis of greatest movement. X and Y keep track of the pen's current position and can only be changed by using the DRAW and MOVE commands. Bresenham's algorithm covers the remainder of the DRAW routine.

Descriptions of Plotting Routines

The routines may be used either as program statements or as interactive commands (hereafter, the term *command* will refer to the statement as either a program statement or an interactive command).

GCLEAR

This routine initializes the graphics package. The *plotter surface* is cleared and the *pen* is set for the lower left-most pixel. The scale is set such that the X axis has a minimum value of zero and the maximum value is 255. The Y axis has a minimum value of zero and the maximum of 191. The lower left hand pixel has the coordinate value of (0,0). The TI BASIC syntax is:

```
CALL LINK("GCLEAR")
```

There are no parameters passed with this command.

SCALE

The SCALE command lets you set the minimum and maximum values of the screen. These values can be real or integer. They may be passed as either numeric expressions or as numeric variables. The minimum *must* be less than the maximum, otherwise, a *BAD ARGUMENT IN . . . type error will be issued. The TI BASIC syntax is:

```
CALL LINK("SCALE",Xmin,Xmax,Ymin,Ymax)
```

The variables may *not* be passed through numeric arrays. The routines do not support arrays at this time. The scaling of the screen may be changed at any time without affecting the data which is already in the screen buffer. When the SCALE command is used, the absolute position of the pen on the screen is not changed. For example, if the pen were in the lower left hand corner of the screen and the scaling were changed, the position of the pen

would *still* be in the lower left-hand corner, no matter what the new scaling.

DRAW

The DRAW command will move the pen from its current location to another point, specified by the user. Since the routine keeps track of the current pen position, all that is necessary is to specify the point to which the pen is to DRAW. The DRAW command is used as follows:

CALL LINK("DRAW",Xposition,Yposition)

The restrictions and freedoms of the parameters for this command are the same as those for the SCALE command. The X and Y positions are the coordinate values of the destination point for the pen. If the position specified is off the screen, the pen will move off the screen, but will only draw the line to the edge of the screen. Although there is no risk of destroying data by drawing off the screen, there are dangers of numeric overflow.

MOVE

This routine performs the same function as the DRAW command, except that the pen is "lifted" before being moved and therefore draws no line. The pen is put back down after reaching its new location. The TI BASIC syntax is:

CALL LINK("MOVE",Xposition,Yposition)

The parameters are the same as the DRAW command.

XAXIS

This command draws a horizontal axis specified by a minimum and maximum along the X axis. The axis will intersect the Y axis at a user specified point. The position of the pen will remain unchanged. Use of this command is as follows:

CALL LINK("XAXIS",Xmin,Xmax,Yintercept)

The parameters may be either numeric expressions or variables. Again, array elements are not allowed.

YAXIS

This command is the Y axis counterpart to the XAXIS command. The TI BASIC syntax is:

CALL LINK("YAXIS",Ymin,Ymax,Xintercept)

LABEL

This command allows the user to output text to the screen, and is capable of outputting the ASCII characters, including upper and lower case. All characters with an ASCII value of less than 32 will be made to look like a 32 (space). All characters with a value greater than 127 will be equated to 127. In this case, this is also a space. Due to the nature of character definitions, the characters will be output to the character blocks, starting at the block designated by the current position of the pen. If the string is contained in a string variable (arrays are not allowed), the TI BASIC syntax is as follows:

CALL LINK("LABEL",variable)

If the user wishes to output the string directly, the TI BASIC syntax is:

CALL LINK("LABEL","string")

GRAPH

This command is used to bring the plots to the screen. The use of GRAPH destroys any BASIC program in memory—so *save your BASIC program before running it*. After the command is invoked, the plot generated will appear on the screen. You can leave this mode by pressing the 'Q' key. The power up screen will appear. If the machine is not turned off, you may add to your graph by returning to TI BASIC and running a new program. The TI BASIC syntax is as follows:

CALL LINK("GRAPH")

This routine requires no parameters.

Loading the Plotting Routines into Mini-Memory

Once the routine has been loaded into the Mini-Memory, it is not necessary to reload it unless the routine has been destroyed for some reason, such as loading some other program into the Mini-Memory. Therefore, once it has been loaded, *do not* invoke the CALL INIT routine! Make sure that no other data is loaded into the Mini-Memory or the lower part of the memory in the Memory Expansion (HEX 2800 to HEX 3000). If the program needs to be loaded into the Mini-Memory, this can be done in one of two ways:

Method 1:

To assemble and load from the program listings—

1. Plug the Editor/Assembler cartridge into the TI-99/4A.
2. Using the Editor, enter the program segment in Listing 1 called BIT1. Save the text entered on a disk file called BIT1.
3. Create disk file BIT2 using Listing 2 as in step 2.
4. Create disk file BIT3 using Listing 3 as in step 2.
5. Create disk file BIT4 using Listing 4 as in step 2.
6. Create disk file SOURCE using Listing 5 as in step 2.
7. Execute the assembler on disk file SOURCE. Direct the object code to disk file GRAPHICS. Use Assembler option R (plus L and S options, if you have a printer).
8. Remove the Editor/Assembler cartridge and insert the Mini-Memory cartridge.
9. Put the disk containing the file GRAPHICS into disk 1.
10. Select the Mini-Memory from the main menu.
11. Select the RE-INITIALIZE option and then press PROCEED.
12. Select the LOAD AND RUN option.
13. Under file name type DSK1.GRAPHICS.
14. When the routine asks for another file name press QUIT.

The routine is now loaded into the Mini-Memory.

[For detailed information on the Editor/Assembler, consult the TI Editor/Assembler manual—Ed.]

Method 2:

To load from the 99'er Magazine-on-Tape cassette, follow the instructions below—

1. Plug Mini-Memory into the TI-99/4A.
2. Select the EASY BUG option from the main menu.
3. Press any key.
4. Press L when a question mark appears.
5. Put the cassette containing the package into the player.
6. Follow the instructions on the screen.
7. When a question mark reappears press QUIT.

The routines need the Memory Expansion for a 6K buffer in addition to the space in the Mini-Memory. If the Memory Expansion is not properly attached a blank screen will appear when the GRAPH routine is invoked.

Example BASIC Programs

The following example will scale the screen so the Xmin is −1, the Xmax is 1, the Ymin is −0.75, and the Ymax is 0.75. The axes will cover the length and height of the screen and intersect at the center. A box will be drawn around the center and the statement "This is a test!" will appear at the bottom of the screen.

```
100 REM **PLOTTING TEST ROUTINE
110 REM
120 REM 99'er Mag.
130 CALL LINK("GCLEAR")
140 A=.75
150 B=1
160 CALL LINK("SCALE",-1,B,-.75,A)
170 CALL LINK("XAXIS",-1,B,0)
180 CALL LINK("YAXIS",-.75,A,0)
190 CALL LINK("MOVE",.5,.375)
200 CALL LINK("DRAW",.5,-.375)
210 C=-.5
220 CALL LINK("DRAW",C,-.375)
230 CALL LINK("DRAW",C,.375)
240 CALL LINK("DRAW",.5,.375)
250 A$="This is a test!"
280 CALL LINK("MOVE",-1,-.75)
290 CALL LINK("LABEL",A$)
300 CALL LINK("GRAPH")
```


ACKNOWLEDGMENT

This package was developed with the computer facilities at the University of Dallas as a project under the guidance of Dr. Bernard Asner and Dr. Carlisle Phillips.

Plotting With The Home Computer

By Joseph G. DeVincentis, Jr.
P. O. Box 375
University of Dallas
Irving, TX 75061

The Assembly Language routines presented in this article will let Home Computer users draw axes, plot curves, and even draw objects in perspective. The software, consisting of plotting routines for the Mini Memory Cartridge, accesses the powerful graphics capabilities of the TI-99/4A through TI BASIC.

The routines supplied in this package require either of the following peripheral configurations:

1. Memory Expansion, cassette recorder, Mini-Memory, and assembled (object file) routines available on this issue's "99'er Magazine-On-Tape." [See page 69]
2. Memory Expansion, disk system, Mini-Memory, Editor/Assembler, and source files (hand entered from listings included with this article.)

```
*************************
*       > BIT1 <        *
* PART ONE OF PLOTTING  *
*       ROUTINES        *
*************************
* BY JOE DEVINCENTIS, JR.
* 99'ER VERSION 2.2.1ALMM
*
       TITL 'BIT MAP LINE'
*
       DEF  DRAW
       DEF  GCLEAR
       DEF  GRAPH
       DEF  LABEL
       DEF  MOVE
       DEF  SCALE
       DEF  XAXIS
       DEF  YAXIS
*
       REF  NUMREF
       REF  STRREF

A      EQU  5
ADR    EQU  0
ARG    EQU  >835C
B      EQU  6
BASE   EQU  >8343
CFI    EQU  >12B8
CHAR   EQU  5
CHRCNT EQU  8
CNT    EQU  9
COLOR  EQU  >1000
COUNT  EQU  9
DELTAA EQU  7
DELTAB EQU  8
DRFLAG EQU  4
ERR    EQU  10
ERRBA  EQU  >1600
ERRBS  EQU  >1700
ERRCOD EQU  >8322
ERRNO  EQU  >1400
ERROR  EQU  >00CE
ERRSNM EQU  >1500
ERRUN  EQU  >2500
FAC    EQU  >834A
FADD   EQU  >0D80
FCOM   EQU  >0D3A
FDIV   EQU  >0FF4
FLAG2  EQU  8
FLAG4  EQU  9
FMUL   EQU  >0E88
FSUB   EQU  >0D7C
GPLWS  EQU  >83E0
KEYCOD EQU  >8375
MASK   EQU  6
MYPGT  EQU  >2800
ORIGIN EQU  >7118
OVF    EQU  >8354
PCT    EQU  >2000
PGT    EQU  >0000
PNT    EQU  >1800
SCAN   EQU  >000E
SNT    EQU  >1B00
STATUS EQU  >837C
VDPRD  EQU  >8800
VDPRST EQU  >8000
VDPWA  EQU  >8C02
VDPWD  EQU  >8C00
XP1    EQU  1
XP2    EQU  3
YP1    EQU  2
YP2    EQU  4
*
       AORG ORIGIN
DELTAX DATA 0
DELTAY DATA 0
SAVR11 DATA 0
STASAV DATA 0
VAL255 DATA >4102,>3700,>0000
       DATA >0000
VAL191 DATA >4101,>5B00,>0000
       DATA >0000
VDPREG DATA >8002,>81E0,>8206
       DATA >83FF,>8403,>8506
       DATA >8717
X      DATA 0
X1     DATA 0
Y      DATA 0
Y1     DATA 0
XSAVE  DATA 0
YSAVE  DATA 0
XDOT   DATA >4001,>0000,>0000
       DATA >0000
YDOT   DATA >4001,>0000,>0000
       DATA >0000
XMAX   DATA >4102,>3700,>0000
       DATA >0000
XMIN   DATA >0000,>0000,>0000
       DATA >0000
YMAX   DATA >4101,>5B00,>0000
       DATA >0000
YMIN   DATA >0000,>0000,>0000
       DATA >0000
*
BUFFER BYTE >FF
       BSS  >FF
       EVEN
CONREG BSS  >20
MREGS  BSS  >20
PREGS  BSS  >20
SREGS  BSS  >20
*
*
DRAW   MOV  @STATUS,@STASAV
       MOV  R11,@SAVR11
       BLWP @DRAW1
       LWPI GPLWS
       MOV  @SAVR11,R11
       MOV  @STASAV,@STATUS
       RT
*
*
GCLEAR MOV  @STATUS,@STASAV
       MOV  R11,@SAVR11
       BLWP @CLEAR
       LWPI GPLWS
       MOV  @SAVR11,R11
       MOV  @STASAV,@STATUS
       RT
*
*
GRAPH  BLWP @VDPSET
       BLWP @GRPAH1
       LWPI GPLWS
SCANIT BL   @SCAN
       MOVB @STATUS,R0
       JNE  SCANIT
       LI   R0,>0020
       MOVB @KEYCOD,R0
       CI   R0,'Q '
       JNE  SCANIT
       LI   R0,VDPRST
       BL   @VADR
       MOV  @2,R11
       B    *R11
*
*
LABEL  MOV  @STATUS,@STASAV
       MOV  R11,@SAVR11
       BLWP @LABEL1
       LWPI GPLWS
       MOV  @SAVR11,R11
       MOV  @STASAV,@STATUS
       RT
*
*
MOVE   MOV  @STATUS,@STASAV
       MOV  R11,@SAVR11
       BLWP @MOVE1
       LWPI GPLWS
       MOV  @SAVR11,R11
       MOV  @STASAV,@STATUS
       RT
*
*
SCALE  MOV  @STATUS,@STASAV
       MOV  R11,@SAVR11
       BLWP @SCALE1
       LWPI GPLWS
       MOV  @SAVR11,R11
       MOV  @STASAV,@STATUS
       RT
*
*
XAXIS  MOV  R11,@SAVR11
       MOV  @STATUS,@STASAV
       LI   R0,MREGS+16
       CLR  *R0+
       CLR  *R0
       BLWP @AXIS
       LWPI GPLWS
       MOV  @STASAV,@STATUS
       MOV  @SAVR11,R11
       RT
*
*
YAXIS  MOV  R11,@SAVR11
       MOV  @STATUS,@STASAV
       LI   R0,MREGS+16
       LI   R1,2
       MOV  R1,*R0+
       SLA  R1,1
       MOV  R1,*R0
       BLWP @AXIS
       LWPI GPLWS
       MOV  @STASAV,@STATUS
       MOV  @SAVR11,R11
       RT
*
*
AXIS   DATA MREGS,AXIS1
*
AXIS1  CLR  R0
       LI   R1,2
       BLWP @NUMREF
       LI   R2,FAC
       LI   R3,ARG
       BL   @TRDATA
       CLR  R0
       LI   R1,1
       BLWP @NUMREF
       LWPI GPLWS
       BL   @FCOM
       LWPI MREGS
       MOVB @STATUS,R0
       ANDI R0,>4000
       JGT  AXCON1
       LI   R0,ERRBA
       B    @ERRSYS
AXCON1 MOV  @X,@XSAVE
       MOV  @Y,@YSAVE
       LI   R2,ARG
       LI   R3,SREGS
       BL   @TRDATA
       MOV  FLAG2,R1
       SRA  R1,1
       AI   R1,1
       BLWP @CONVTR
       MOV  R0,@X(FLAG
       LI   R2,SREGS
       LI   R3,FAC
       BL   @TRDATA
       MOV  FLAG2,R1
       SRA  R1,1
       AI   R1,1
       BLWP @CONVTR
       MOV  R0,@X1(FLA
       S    @X(FLAG4),
       MOV  R0,@DELTAX
       CLR  R0
       LI   R1,3
       BLWP @NUMREF
       LI   R1,2
       MOV  FLAG4,FLAG
       JEQ  AXCON2
       SRA  R1,1
AXCON2 BLWP @CONVTR
       NEG  FLAG2
       NEG  FLAG4
       MOV  R0,@Y(FLAG
       MOV  R0,@Y1(FLA
       CLR  @DELTAY(FL
       BLWP @DRAW3
       MOV  @XSAVE,@X
       MOV  @YSAVE,@Y
       RTWP
*
*
       PAGE
```

```
**********************
*       > BIT2 <     *
* PART TWO OF PLOTTING *
*       ROUTINES     *
**********************
* 99'ER VERSION 2.2.1ALMM
*
*LINE 0002
CLEAR  DATA MREGS,CLEAR1
CLEAR1 LI   R1,MYPGT
       LI   R2,>1800
CLCON1 CLR  *R1
       INCT R1
       DECT R2
       JNE  CLCON1
       CLR  @X
       CLR  @Y
       LI   R2,VAL255
       LI   R3,XMAX
       BL   @TRDATA
       LI   R2,VAL191
       LI   R3,YMAX
       BL   @TRDATA
       LI   R3,>4001
       MOV  R3,@XDOT
       MOV  R3,@YDOT
       CLR  @XMIN
       CLR  @YMIN
       LI   R1,3
       LI   R2,2
CLCON2 CLR  @XDOT(R2)
       CLR  @YDOT(R2)
       CLR  @XMIN(R2)
       CLR  @YMIN(R2)
       INCT R2
       DEC  R1
       JNE  CLCON2
       RTWP
*
*
CONVTR DATA CONREG,CONVT1
*
CONVT1 MOV  @2(R13),R5
       CI   R5,1
       JNE  CVCON1
       CLR  R5
       CLR  R6
       JMP  CVCON2
CVCON1 LI   R5,16
       LI   R6,8
CVCON2 LI   R2,FAC
       LI   R3,ARG
       BL   @TRDATA
       LI   R2,XMIN
       A    R5,R2
       LI   R3,FAC
       BL   @TRDATA
       LWPI GPLWS
       BL   @FSUB
       LWPI CONREG
       LI   R2,FAC
       LI   R3,ARG
       BL   @TRDATA
       LI   R2,XDOT
       A    R6,R2
       LI   R3,FAC
       BL   @TRDATA
       LWPI GPLWS
       BL   @FDIV
       BL   @CFI
       LWPI CONREG
       MOV  @FAC,*R13
       RTWP
*
*
DRAW1  DATA MREGS,DRAW2
DRAW3  DATA SREGS,DRAW4
*
DRAW2  CLR  R0
       LI   R1,1
       BLWP @NUMREF
       BLWP @CONVTR
       MOV  R0,@X1
       S    @X,R0
       MOV  R0,@DELTAX
       CLR  R0
       LI   R1,2
       BLWP @NUMREF
       BLWP @CONVTR
       MOV  R0,@Y1
       S    @Y,R0
       MOV  R0,@DELTAY
```

```
DRAW4  ABS  @DELTAX
       ABS  @DELTAY
       C    @DELTAY,@DELTAX
       JGT  DRCON1
       MOV  @DELTAX,DELTAA
       MOV  @DELTAY,DELTAB
       MOV  @X,A
       MOV  @Y,B
       CLR  DRFLAG
       JMP  DRCON2
DRCON1 MOV  @DELTAX,DELTAB
       MOV  @DELTAY,DELTAA
       MOV  @X,B
       MOV  @Y,A
       SETO DRFLAG
DRCON2 C    @X,@X1
       JLT  DRCON3
       SETO R2
       JMP  DRCON4
DRCON3 LI   R2,1
DRCON4 C    @Y,@Y1
       JLT  DRCON5
       SETO R3
       JMP  DRCON6
DRCON5 LI   R3,1
DRCON6 MOV  DELTAA,COUNT
       INC  COUNT
       MOV  DELTAB,R0
       SLA  R0,1
       MOV  DELTAA,R1
       S    R1,R0
       MOV  R0,ERR
       S    R1,R0
       MOV  DELTAB,R1
       SLA  R1,1
REDRW1 MOV  DRFLAG,DRFLAG
       JNE  REDRW2
       MOV  A,@X
       MOV  B,@Y
       JMP  REDRW3
REDRW2 MOV  A,@Y
       MOV  B,@X
REDRW3 BLWP @PLOT
       MOV  ERR,ERR
       JGT  CHNG
       A    R1,ERR
       JMP  INCR
CHNG   MOV  DRFLAG,DRFLAG
       JNE  REDRW4
       A    R3,B
       JMP  REDRW5
REDRW4 A    R2,B
REDRW5 A    R0,ERR
INCR   MOV  DRFLAG,DRFLAG
       JNE  REDRW6
       A    R2,A
       JMP  REDRW7
REDRW6 A    R3,A
REDRW7 DEC  COUNT
       JNE  REDRW1
       MOV  @X1,@X
       MOV  @Y1,@Y
       RTWP
*
*
ERRSYS MOV  R0,@ERRCOD
       LWPI GPLWS
       LI   R11,>000E
       MOV  *R11,R11
       B    @ERROR
*
*
GRPAH1 DATA MREGS,GRPAH2
*
GRPAH2 CLR  R0
       BL   @VADW
       LI   R1,>1800
       LI   R2,MYPGT
GRCON1 MOVB *R2+,@VDPWD
       DEC  R1
       JNE  GRCON1
       RTWP
*
*
LABEL1 DATA MREGS,LABEL2
*
LABEL2 CLR  R0
       LI   R1,1
       LI   R2,>FF00
       MOVB R2,@BUFFER
       LI   R2,BUFFER
```

```
       BLWP @STRREF
       MOV  @X,XP1
       CI   XP1,256
       JLT  LBCON1
       RTWP
LBCON1 CI   XP1,>8000
       JL   LBCON2
       RTWP
LBCON2 MOV  @Y,YP1
       CI   YP1,192
       JLT  LBCON3
       RTWP
LBCON3 CI   YP1,>8000
       JL   LBCON4
       RTWP
LBCON4 CLR  CHRCNT
       MOVB @BUFFER,CHRCNT
       SWPB CHRCNT
       NEG  YP1
       AI   YP1,191
       SRA  XP1,3
       SRA  YP1,3
       SLA  YP1,5
       A    XP1,YP1
       MOV  YP1,ADR
       A    CHRCNT,YP1
       CI   YP1,768
       JL   LBCON5
       LI   R6,768
       S    R6,YP1
       MOV  YP1,CHRCNT
LBCON5 LI   R7,BUFFER+1
       SLA  ADR,3
       AI   ADR,MYPGT
LLOOP1 CLR  CHAR
       MOVB *R7+,CHAR
       SWPB CHAR
       CI   CHAR,32
       JHE  LBCON6
       CLR  CHAR
       JMP  LBCON8
LBCON6 CI   CHAR,127
       JLE  LBCON7
       LI   CHAR,127
LBCON7 AI   CHAR,-32
       SLA  CHAR,3
LBCON8 AI   CHAR,CHRTBL
       LI   CNT,8
LLOOP2 SOC  *CHAR+,*ADR+
       CI   ADR,MYPGT+>1801
       JL   LBCON9
       RTWP
LBCON9 DECT CNT
       JNE  LLOOP2
       DEC  CHRCNT
       JNE  LLOOP1
       LI   R0,>FF00
       MOVB R0,@BUFFER
       RTWP
*
*
MOVE1  DATA MREGS,MOVE2
*
MOVE2  CLR  R0
       LI   R1,1
       BLWP @NUMREF
       BLWP @CONVTR
       MOV  R0,@X
       CLR  R0
       LI   R1,2
       BLWP @NUMREF
       BLWP @CONVTR
       MOV  R0,@Y
       RTWP
*
*
       PAGE
```

```
******************************
*         > BIT 3<           *
* PART THREE OF PLOTTING *
*        ROUTINES            *
******************************
* 99'ER VERSION 2.2.1ALMM
*
*LINE 0002
*
PLOT    DATA PREGS,PLOT1
*
PLOT1   MOV  @X,XP1
        CI   XP1,256
        JLT  PLCON1
        RTWP
PLCON1  CI   XP1,>8000
        JL   PLCON2
        RTWP
PLCON2  MOV  @Y,YP1
        CI   YP1,192
        JLT  PLCON3
        RTWP
PLCON3  CI   YP1,>8000
        JL   PLCON4
        RTWP
PLCON4  MOV  XP1,XP2
        NEG  YP1
        AI   YP1,191
        MOV  YP1,YP2
        SRA  YP1,3
        SLA  YP1,5
        SRA  XP1,3
        MOV  YP1,ADR
        A    XP1,ADR
        SLA  ADR,3
        ANDI YP2,>0007
        A    YP2,ADR
        AI   ADR,MYPGT
        MOVB *ADR,CHAR
        ANDI XP2,>0007
        MOV  ADR,R4
        MOV  XP2,R0
        LI   MASK,>8000
        SRC  MASK,0
        MOV  R4,ADR
        SOC  MASK,CHAR
        MOVB CHAR,*ADR
        RTWP
*
*
SCALE1  DATA MREGS,SCALE2
*
SCALE2  CLR  R5
        CLR  R6
        BL   @NUMGET
        LI   R5,2
        LI   R6,16
        BL   @NUMGET
        CLR  R5
        CLR  R6
        BL   @NUMSUB
        LI   R5,8
        LI   R6,16
        BL   @NUMSUB
        CLR  R5
        BL   @NUMDIV
        LI   R5,8
        BL   @NUMDIV
        RTWP
*
NUMDIV  MOV  R11,R10
        LI   R2,VAL255
        A    R5,R2
        LI   R3,FAC
        BL   @TRDATA
        LI   R2,XDOT
        A    R5,R2
        LI   R3,ARG
        BL   @TRDATA
        LWPI GPLWS
        BL   @FDIV
        LWPI MREGS
        LI   R2,FAC
        LI   R3,XDOT
        A    R5,R3
        BL   @TRDATA
        B    *R10
*
NUMGET  MOV  R11,R10
        CLR  R0
        LI   R1,2
        A    R5,R1
        LI   R2,FAC
        LI   R3,XMAX
        A    R6,R3
        BLWP @NUMREF
        BL   @TRDATA
        DEC  R1
        LI   R2,FAC
        LI   R3,XMIN
        A    R6,R3
        BLWP @NUMREF
        BL   @TRDATA
        LI   R2,XMAX
        A    R6,R2
        LI   R3,ARG
        BL   @TRDATA
        LWPI GPLWS
        BL   @FCOM
        LWPI MREGS
        MOVB @STATUS,R0
        ANDI R0,>4000
        JGT  SCCON1
        LI   R0,ERRBA
        B    @ERRSYS
SCCON1  B    *R10
*
NUMSUB  MOV  R11,R10
        LI   R2,XMIN
        A    R6,R2
        LI   R3,FAC
        BL   @TRDATA
        LI   R2,XMAX
        A    R6,R2
        LI   R3,ARG
        BL   @TRDATA
        LWPI GPLWS
        BL   @FSUB
        LWPI MREGS
        LI   R9,>0100
        CB   R9,@OVF
        JNE  SCCON2
        LI   R0,ERRNO
        B    @ERRSYS
SCCON2  CLR  R7
        CLR  R8
        MOVB @FAC,R7
        SWPB @FAC
        MOVB @FAC,R8
        ABS  R7
        ABS  R8
        MOVB R8,@FAC
        SWPB @FAC
        MOVB R7,@FAC
        LI   R2,FAC
        LI   R3,XDOT
        A    R5,R3
        BL   @TRDATA
        B    *R10
*
*
TRDATA  MOV  *R2+,*R3+
        MOV  *R2+,*R3+
        MOV  *R2+,*R3+
        MOV  *R2,*R3
        RT
*
*
VADW    ORI  R0,>4000
VADR    SWPB R0
        MOVB R0,@VDPWA
        SWPB R0
        MOVB R0,@VDPWA
        RT
*
*
VDPSET  DATA MREGS,VDP1
*
VDP1    LI   R2,VDPREG
        LI   R1,7
VDCON1  MOV  *R2+,R0
        BL   @VADR
        DEC  R1
        JNE  VDCON1
        LI   R0,SNT
        LI   R1,>D000
        BL   @VADW
        MOVB R1,@VDPWD
        LI   R0,PCT
        BL   @VADW
        LI   R1,COLOR
        LI   R2,>1800
VDCON2  MOVB R1,@VDPWD
        DEC  R2
        JNE  VDCON2
        LI   R0,PNT
        BL   @VADW
        LI   R3,3
VDCON3  CLR  R1
        LI   R2,256
VDCON4  MOVB R1,@VDPWD
        AI   R1,>0100
        DEC  R2
        JNE  VDCON4
        DEC  R3
        JNE  VDCON3
        RTWP
*
*
*
        PAGE
```

```
**************************
*        > BIT4 <        *
* PART FOUR OF PLOTTING  *
*        ROUTINES        *
**************************
* 99'ER VERSION 2.2.1ALMM
*
*LINE 0002
*
CHRTBL DATA >0000,>0000,>0000,>0000  blank
       DATA >0010,>1010,>1010,>0010  !
       DATA >0028,>2828,>0000,>0000  "
       DATA >0028,>287C,>287C,>2828  #
       DATA >0038,>5450,>3814,>5438  $
       DATA >0060,>6408,>1020,>4C0C  %
       DATA >0020,>5050,>2054,>4834  &
       DATA >0008,>0810,>0000,>0000  '
       DATA >0008,>1020,>2020,>1008  (
       DATA >0020,>1008,>0808,>1020  )
       DATA >0000,>2810,>7C10,>2800  *
       DATA >0000,>1010,>7C10,>1000  +
       DATA >0000,>0000,>0030,>1020  ,
       DATA >0000,>0000,>7C00,>0000  -
       DATA >0000,>0000,>0000,>3030  .
       DATA >0000,>0408,>1020,>4000  /
       DATA >0038,>4444,>4444,>4438  0
       DATA >0010,>3010,>1010,>1038  1
       DATA >0038,>4404,>0810,>207C  2
       DATA >0038,>4404,>1804,>4438  3
       DATA >0008,>1828,>487C,>0808  4
       DATA >007C,>4078,>0404,>4438  5
       DATA >0018,>2040,>7844,>4438  6
       DATA >007C,>0408,>1020,>2020  7
       DATA >0038,>4444,>3844,>4438  8
       DATA >0038,>4444,>3C04,>0830  9
       DATA >0000,>3030,>0030,>3000  :
       DATA >0000,>3030,>0030,>1020  ;
```

```
       DATA >0008,>1020,>4020,>1008  <
       DATA >0000,>007C,>007C,>0000  =
       DATA >0020,>1008,>0408,>1020  >
       DATA >0038,>4404,>0810,>0010  ?
       DATA >0038,>445C,>545C,>4038  @
       DATA >0038,>4444,>7C44,>4444  A
       DATA >0078,>2424,>3824,>2478  B
       DATA >0038,>4440,>4040,>4438  C
       DATA >0078,>2424,>2424,>2478  D
       DATA >007C,>4040,>7840,>407C  E
       DATA >007C,>4040,>7840,>4040  F
       DATA >003C,>4040,>5C44,>4438  G
       DATA >0044,>4444,>7C44,>4444  H
       DATA >0038,>1010,>1010,>1038  I
       DATA >0004,>0404,>0404,>4438  J
       DATA >0044,>4850,>6050,>4844  K
       DATA >0040,>4040,>4040,>407C  L
       DATA >0044,>6C54,>5444,>4444  M
       DATA >0044,>6464,>544C,>4C44  N
       DATA >007C,>4444,>4444,>447C  O
       DATA >0078,>4444,>7840,>4040  P
       DATA >0038,>4444,>4454,>4834  Q
       DATA >0078,>4444,>7850,>4844  R
       DATA >0038,>4440,>3804,>4438  S
       DATA >007C,>1010,>1010,>1010  T
       DATA >0044,>4444,>4444,>4438  U
       DATA >0044,>4444,>2828,>1010  V
       DATA >0044,>4444,>5454,>5428  W
       DATA >0044,>4428,>1028,>4444  X
       DATA >0044,>4428,>1010,>1010  Y
       DATA >007C,>0408,>1020,>407C  Z
       DATA >0038,>2020,>2020,>2038  [
       DATA >0000,>4020,>1008,>0400  \
       DATA >0038,>0808,>0808,>0838  ]
       DATA >0000,>1028,>4400,>0000  ^
```

```
       DATA >0000,>0000,>0000,>7C00  _
       DATA >0000,>2010,>0800,>0000  `
       DATA >0000,>0038,>447C,>4444  a
       DATA >0000,>0078,>2438,>2478  b
       DATA >0000,>003C,>4040,>403C  c
       DATA >0000,>0078,>2424,>2478  d
       DATA >0000,>007C,>4078,>407C  e
       DATA >0000,>007C,>4078,>4040  f
       DATA >0000,>003C,>405C,>4438  g
       DATA >0000,>0044,>447C,>4444  h
       DATA >0000,>0038,>1010,>1038  i
       DATA >0000,>0008,>0808,>4830  j
       DATA >0000,>0024,>2830,>2824  k
       DATA >0000,>0040,>4040,>407C  l
       DATA >0000,>0044,>6C54,>4444  m
       DATA >0000,>0044,>6454,>4C44  n
       DATA >0000,>007C,>4444,>447C  o
       DATA >0000,>0078,>4478,>4040  p
       DATA >0000,>0038,>4454,>4834  q
       DATA >0000,>0078,>4478,>4844  r
       DATA >0000,>003C,>4038,>0478  s
       DATA >0000,>007C,>1010,>1010  t
       DATA >0000,>0044,>4444,>4438  u
       DATA >0000,>0044,>4428,>2810  v
       DATA >0000,>0044,>4454,>5428  w
       DATA >0000,>0044,>2810,>2844  x
       DATA >0000,>0044,>2810,>1010  y
       DATA >0000,>007C,>0810,>207C  z
       DATA >0018,>2020,>4020,>2018  {
       DATA >0010,>1010,>0010,>1010  |
       DATA >0030,>0808,>0408,>0830  }
       DATA >0000,>2054,>0800,>0000  ~
       DATA >0000,>0000,>0000,>0000  del
*
*
       END
```

```
**************************
*       > SOURCE <       *
*                        *
* PART FIVE OF PLOTTING  *
*        ROUTINES        *
**************************
*
*99'ER VERSION 2.2.1ALMM
*
       COPY "DSK1.BIT1"
       COPY "DSK1.BIT2"
       COPY "DSK1.BIT3"
       COPY "DSK1.BIT4"
       END
```

IT'S SUPER LANGUAGE!

By John Clulow
Technical Editor

MAGIC CRAYON

Learning Assembly Language The Hard Way

Part 2 of Patricia Swift's article, **A Screen Printing Utility**, ***has been postponed until next issue because of the CES announcement that the Epson MX-80 printer (with dot-addressable graphics installed) would be TI's new 99/4 matrix printer. Extra article development time is needed to ensure screen-dump compatibility with both the MX-100 and TI's version of the MX-80. We think you'll learn quite a lot from this issue's substitute article in the interim.***

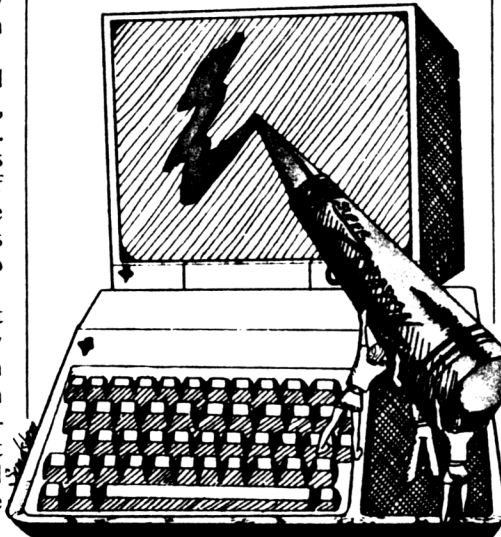
Like many other 99'ers, I was anxious to receive the long awaited Editor/Assembler package. When it finally arrived, I remember the excitement of unwrapping the 470 page manual—and the sinking feeling when I read, "This manual assumes that you already know a programming language, preferably an assembly language."

My anxiety grew as I thumbed through it—there were no pictures, cartoons, or fill-in-the-blank examples. It did say, "There are many fine books available which teach the basics of assembly language." So I called the local computer stores. The only books they were aware of, however, also assumed familiarity with the basics.

I guess I had some fuzzy ideas about assembly language in the back of my mind; it was qualitatively different from higher level languages, requiring an in-depth knowledge of digital electronics and a capacity for the most detailed sort of logico-mathematical thought. In short—nothing seemed more difficult . . .

But my experience thus far seemed to confirm my worst fear. Learning assembly language presumed a prior knowledge of assembly language; it was not merely difficult, it was *impossible*. After running *Tombstone City* a few times and typing in Pat Swift's *Life* program (in Vol. 1, No. 4), I put the Editor/Assembler on a shelf thinking maybe I'd learn about it gradually over the next year or two.

It would still be there gathering dust were it not for a back injury that kept me flat on the floor, unable to do anything *except* read the manual. I was surprised to discover that writing an assembly language program is similar to, and in some respects simpler than writing a program in BASIC. A new programming context or conceptual model is required. But to get started, I found that this picture could be primitive, containing many over-simplifications and approximations.

The picture I developed enabled me to successfully formulate and execute a simple programming objective. The program and associated underlying concepts are presented here to facilitate the learning process for others who, like me, find it hard to overcome preconceived notions about how difficult assembly language is. The program should not be taken as a model of exemplary programming technique; at this point my conception of "good programming" is programming that works . . . period. You will undoubtedly be able to find ways to improve this one—to make it work faster and utilize memory more efficiently—and in so doing, further develop the concepts presented.

In assembly language, four video display modes are available: Graphics (or Pattern) Mode, Text Mode, Bit-Map Mode (99/4A only), and Multicolor Mode. In Multicolor Mode, the screen is divided into a grid of 64 x 48, with each box measuring 4 pixels on a side. Each box can have a color assigned to it. The program allows use of a joystick to move a flashing cursor on the screen. Whenever the fire button is depressed, the cursor leaves a trail of small, colored boxes. The following single key commands are available:

C – *Change Color*. Displays a color palette and pointer. Move the pointer to the desired color with the joystick. Press the fire button to make that the color of the boxes, or press the C key to make it the color of the screen background.

S – *Save Screen*. Saves the current contents of the screen as "DSK1.SCREEN".

R – *Recall Screen*. Loads the contents of "DSK1.SCREEN" for subsequent modification.

E – *Erase Screen*. Erases the screen contents.

T – *Terminate*. Returns to the Master Title Screen.

In order to understand how the program works, it will be helpful to differentiate two systems. You probably know that the Central Processing Unit (CPU) in the Home Computer is the TMS9900. It has three built-in, 16-bit "hardware" registers (the Program Counter, Workspace Pointer, and Status Register) and makes use of sixteen workspace registers located in read-write memory. Because these 16-bit workspace registers are not located on the chip, they are called "software" registers. The CPU can directly address the read-write memory (RAM) in the Memory Expansion Unit and CPU scratch pad, as well as ROM in the console, Command Modules, and various peripherals. However, it cannot directly address the 16K of RAM built into the console.

That 16K RAM block is addressed by another microprocessor—the TMS9918

Continued on p. 60

Crayon . . . from p. 57

Table 1 VDP RAM MEMORY —Editor/Assembler—

Address of First Byte Decimal	Hex	Length of Block, Bytes	Contents
0	>0000	768	Screen Image Table
768	>0300	128	Sprite Attribute List
896	>0380	128	Color Table
1024	>0400	896	Sprite Descriptor Table
1920	>0780	128	Sprite Motion Table
2048	>0800	2048	Pattern Descriptor Table and Peripheral Access Blocks
4096	>1000	10199	More Peripheral Access Blocks and Buffers
14295	>37D7	2089	Reserved for Diskette Device Service Routines
16383	>3FFF	——	Last Address
Total 16384 Bytes			

(or 9918A if you have a 99/4A). This Video Display Processor (VDP) has eight 8-bit hardware registers and four 8-bit software registers. The software registers are located in read-write memory locations which can also be addressed by the CPU. The fact that these four bytes can be addressed by both the CPU and VDP makes it possible for the CPU and VDP systems to transfer data back and forth. The CPU addresses of the registers— > 8800, > 8802, > 8C00, > 8C02—are assigned respectively to the symbols VDPRD (VDP Read Data), VDPSTA (VDP Status), VDPWD (VDP Write Data), and VDPWA (VDP Write Address).

We don't have to be concerned with the details of moving data to and from VDP RAM and to VDP registers, however, thanks to some of the built-in programs called "utilities." The five utilities of use are identified by the symbols VSBW, VMBW, VSBR, VMBR, and VWTR. The respective functions of these programs are VDP RAM: Single Byte Write, Multiple Byte Write, Single Byte Read, Multiple Byte Read, and Write to Register. User workspace registers are used to pass parameters—e.g., the number of bytes to read or write—to the utility.

The standard utilization of VDP RAM in the Editor/Assembler is shown on Table 1. The blocks involved in the multicolor mode are the Screen Image and Pattern Descriptor Tables. Before entering multicolor mode, the Screen Image Table is initialized. The 768 bytes of the table are divided into six 128-byte sets. Each set is further subdivided into four 32-byte groups. To initialize the table, the numbers 0-31 are written in order into each of the four 32 byte groups in the first set: 0, 1, 2, . . . 31 four times. Then the numbers 32-61 are written four times into the next 128-byte set. This process is continued until the numbers 160-191 are written four times in the sixth 128-byte set. In my program, I didn't want this process to be visible on the screen, so I first put the display in Text Mode and made the foreground and background colors gray.

Once the Screen Image Table is initialized, color boxes are placed on the screen by means of the Pattern Descriptor Table. Each 4 x 4 pixel box on the screen corresponds to half a byte in the Pattern Descriptor Table. To place a colored box on the screen, the appropriate color code is written in the nybble (4 bits) in the Pattern Descriptor Table which corresponds to the desired screen position.

The first eight bytes of the Pattern Descriptor Table correspond to boxes in a column beginning in the upper left corner of the screen. The first four bits in byte #1 contain the color of the box in the extreme upper left corner and the last four bits the color of the box immediately to the right of the first box. Byte #2 contains the colors of the two boxes immediately under the first two, and so on for the first eight bytes.

The ninth byte in the table contains the colors for the pair of boxes in a new column beginning again at the top of the screen. Subsequent bytes follow this pattern corresponding to 32 columns of box pairs with eight pairs in each column. This group of 256 bytes thus takes care of the top sixth of the screen.

The 257th byte corresponds to the beginning of a new column of box pairs starting again on the left side of the screen. The six 256-byte groups thus correspond to the 3,072 possible boxes in multicolor mode. [Since the color of each box is indicated in a name table in memory, and the names are mapped onto the screen according to their position in the table, this multicolor mode is a *true* memory-mapped configuration. It does, however, trade off lower resolution for color memory-mapping capability, but the high-resolution sprites

are still available. For an explanation of sprites and an introduction to the high-resolution bit-map mode, see *3-D Animation With the TMS9918A Video Chip* in this issue—Ed.]

In the program, a double-size sprite provides a reference point for determining where boxes will appear. The dot row and dot column of the sprite can be determined at any time by referring to the Sprite Attribute List in VDP RAM. Then, since boxes are supposed to appear in the center of the sprite, the screen location can be calculated by adding 8 to the dot row and dot column, which represent the sprite's upper left corner. But in order to find the corresponding location in the Pattern Descriptor Table, a few more calculations must be performed.

If we let R and C be the dot row and dot column desired for the box location, the number of complete 256-byte groups above that location is the integer quotient of R/32. Multiplying that number by 256 thus gives the first component of the offset in the Pattern Descriptor Table.

Similarly, the integer quotient of C/8 gives the number of complete 8-byte columns to the left of the location. So that number is multiplied by 8 and added to the offset. Dividing the remainder of R/32 by 4 gives the number of bytes above the location in the 8-byte column the location is in. Adding that to the offset gives the offset for the byte in the Pattern Descriptor Table.

But we still have to know if the desired location is the most or least significant nybble of the byte, and to determine that we can divide the remainder of C/8 by 4. If the integer quotient is 0, it's the left nybble; if 1, it's the right nybble. The appropriate color code then need only be placed in the correct nybble (leaving the other one unchanged) and the box appears just where it should.

Let's consider an example: Suppose the upper left corner of the sprite were at dot row 83 and dot column 147. The center of the sprite would then be at 91 and 155. The number of complete groups (32 columns with 8 bytes in each) above that location is 2—i.e., INT(91/32). So the initial component of the offset is 2 * 256 or 512 bytes. The number of 8-byte columns to the left of the location is INT(155/8) or 19. That makes the offset 531. Above the location, in its 8-byte column, there are 6 bytes—i.e., INT((remainder 91/32)/4)—giving an offset of 537. The remainder of 155/8 is 3, and INT(3/4) is 0, so the nybble of interest is the most significant (left) one of the 538th byte of the Pattern Descriptor Table.

Now let's take a brief look at the source listing. The first section consists of a number of assembler directives. The DEF directive makes the symbol MARKER available to other programs, and the REF directives make several utilities available for use in MARKER. Then there are a variety of other assembler directives. The simplest type is EQUate which assigns a constant to a symbol at assembly time. USRWS, for instance, will be assigned the value of >20BA(8378), and that value replaces the symbol wherever it appears in an operand; the label may subsequently be substituted for the number.

The mnemonic BSS stands for Block Starting with Symbol and this directive causes the assembler to advance its location counter without writing anything into the object program. It leaves an empty area (of the number of bytes specified in the operand) which can then be used as a storage space for data later on. The label is set equal to the memory location of the first byte in the block at the time the object program is loaded. (Since this program is relocatable, the place where the loader program decides to start loading it may change depending on what other programs have already been loaded.)

The DATA, BYTE, and TEXT directives are similar to BSS except that the contents of the buffer are explicitly

Continued on p. 83

in a DATA statement. INTERNAL format saves the data in the same way that the computer stores the information internally. Numbers require 8 characters (bytes). Strings (i.e., names) require 1 byte (for the length) plus the data itself. I usually save my data in INTERNAL format so that I know the length needed for numbers no matter how big or small they are.

The BASICs of Record Keeping

Let's write a part of a program that will save each bowler's name, his pin average and his handicap. Pretend that we have 60 bowlers in our league. If we restrict each bowler's name to a maximum of 47 characters, we will need a total of 64 bytes per bowler (47 bytes + 1 = 48 for the name + 8 for the average + 8 for the handicap = 64). We can therefore fit the data for 3 bowlers into one 192 byte record. (see figure 1) If you have filled up a record by the time the program hits the CLOSE statement, TI BASIC will fill the record with blanks and write it out. You do not have to worry about writing out a last record that is partially full. Just remember to always code in a CLOSE statement. To read the data file into your program, you need code that almost duplicates the write code. (see figure 2)

Continued on p. 84

```
090 REM ROOM FOR 60 BOWLERS NAMES, AVERAGES, HANDICAPS
100 DIMENSION B_NAME(60),B_AVG(60),B_HANDI(60)

995 REM OPEN THE FILE FOR OUTPUT
1000 OPEN #1:"CS1",OUTPUT,INTERNAL,SEQUENTIAL,FIXED 192
1010 X=1
1020 FOR I=1 TO 60
1025 REM SEE IF RECORD IS FULL
1030 IF X=3 THEN 1100
1040 X=X+1
1050 REM ADD TO RECORD- BUT DON'T WRITE IT OUT
1060 PRINT #1:B_NAME(I);B_AVG(I);B_HANDI(I);
1070 GOTO 1120
1090 REM ADD TO RECORD AND WRITE IT OUT!
1100 PRINT #1:B_NAME(I);B_AVG(I);B_HANDI(I)
1110 X=1
1120 NEXT I
1130 CLOSE #1
```

Figure 1

```
195 REM OPEN THE FILE FOR INPUT
200 OPEN #1:"CS1",INPUT,INTERNAL,SEQUENTIAL,FIXED 192
210 X=1
220 FOR I=1 TO 60
230 REM SEE IF RECORD IS FULL
240 IF X=3 THEN 300
250 X=X+1
260 REM READ RECORD- BUT DON'T READ TAPE
270 INPUT #1:B_NAME(I);B_AVG(I);B_HANDI(I);
280 GOTO 320
290 REM READ RECORD AND GET NEXT TAPE
300 INPUT #1:B_NAME(I);B_AVG(I);B_HANDI(I)
310 X=1
320 NEXT I
330 CLOSE #1
```

Figure 2

Crayon . . . from p. 61

defined in the operand field. The label is assigned the address of the first byte at the time the object program is loaded. All of these buffer areas are contiguous. For example, look at the instructions immediately after the label MARKER. The pattern codes for two double-size sprites, the cursor and arrow, are loaded into the Sprite Descriptor Table in VDP RAM. Since the pattern data for ARROW is contiguous with that of CURSOR in both CPU and VDP RAM, all 64 bytes can be loaded in one shot.

You should have little trouble figuring out the rest of the program by reading the comments provided and referring to the manual. But don't stop after you understand how it works —try to make some changes. To start with, try changing the shape and colors of the sprite cursor, the arrangement of the color palette on the screen, etc. Then try to make the program more efficient in speed and utilization of memory.

Be prepared to run into problems; it's through encountering and solving them that you'll learn most rapidly. When I decided to stop reading and start trying to write a program, I had visions of seeing a curl of white smoke rise from the computer's cooling vents, but that didn't happen to me, and probably won't happen to you either. So don't be afraid to experiment.

MAGIC CRAYON
99'er Version 1.6.1 AL

```
x       DEF  MARKER
        REF  VSBW,VMBW,VMBR,VSBR
        REF  VWTR,KSCAN,DSRLNK
*
* DEFINITION OF LABELS
*
SCREEN BSS  >300
PALET  BSS  >600
PATRN  BSS  >600
ROW    BSS  1
COL    BSS  1
CURSOR DATA >8040,>2010,>0804,>0000
       DATA >0000,>0408,>1020,>4080
       DATA >0102,>0408,>1020,>0000
       DATA >0000,>2010,>0804,>0201
ARROW  DATA >0102,>0408,>0000,>0000
       DATA >0000,>0000,>0000,>0000
       DATA >0080,>4020,>0000,>0000
       DATA >0000,>0000,>0000,>0000
ATTRIB DATA >5878,>800F,>D000
ARRATT DATA >6578,>8401
PDATA  DATA >0600,>1000,>0000,>0600
       DATA >000B
       TEXT 'DSK1.SCREEN'
ZERO   DATA >0000
D32    DATA >0020
D8     DATA >0008
GRAY   DATA >EEEE
MAX    DATA >05FF
COLMAX DATA >0100
LOAD   BYTE >05
BLACK  BYTE >11
ONE    BYTE >01
TWO    BYTE >02
FCOLOR BYTE >10
BCOLOR BYTE >0E
H18    BYTE >12
H14    BYTE >0E
H11    BYTE >0B
H07    BYTE >07
H06    BYTE >06
H05    BYTE >05
H02    BYTE >02
NOKEY  BYTE >FF
PAB    EQU  >0F80
USRWS  EQU  >20BA
PNTR   EQU  >8356
UNIT   EQU  >8374
FIRE   EQU  >8375
JOYSTY EQU  >8376
JOYSTX EQU  >8377
SPRITE EQU  >837A
STATUS EQU  >837C
GPLWS  EQU  >83E0
*
* DEFINE SPRITE PATTERNS FOR CHRS 128 AND 132
*
MARKER LWPI USRWS          LOAD WORKSPACE POINTER / START
       LI   R0,>400        VDP ADDRESS CH 128 SPRITE DESCRIPTOR TABLE
       LI   R1,CURSOR      CPU ADDRESS OF CHAR PATTERN
       LI   R2,64          64 BYTES TO MOVE (2 PATERNS)
       BLWP @VMBW          LOAD DATA TO VDP RAM
*
* SET FOREGROUND AND BACKGROUND TO GRAY
*
       LI   R0,>01F0       PLACE IN TEXT MODE
       BLWP @VWTR          WRITE TO VDP R1
       LI   R0,>07EE       SET FORE AND BACKGROUND TO GRAY
       BLWP @VWTR          WRITE TO VDP R7
*
* INITIALIZE SCREEN IMAGE TABLE FOR MULTICOLOR MODE
*
       LI   R0,SCREEN      INITIALIZE POINTER
       LI   R1,6           INITIALIZE GROUP COUNTER
       CLR  R2             INITIALIZE VALUE
LOOP0  LI   R3,4           INITIALIZE REPETITIONS COUNTER
LOOP1  LI   R4,>20         INITIALIZE VALUE COUNTER
       MOVB R2,R5          START REPETITION
LOOP2  MOVB R5,*R0+        STORE VALUE IN ARRAY SCREEN
       AI   R5,>0100       CHANGE TO NEXT VALUE
       DEC  R4             COUNT DOWN FOR NEXT VALUE
       JNE  LOOP2          DO NEXT VALUE
       DEC  R3             DEC REPETITION COUNTER
       JNE  LOOP1          DO NEXT REPETITION
       AI   R2,>2000       NEXT STARTING VALUE
       DEC  R1             DEC GROUP COUNTER
       JNE  LOOP0          DO NEXT GROUP
       LI   R0,>00         VDP ADDRESS FOR SCREEN IMAGE
       LI   R1,SCREEN      CPU ADDRESS OF DATA BUFFER
       LI   R2,>300        768 BYTES TO WRITE
       BLWP @VMBW          INITIALIZE VDP SCREEN IMAGE
*
* INITIALIZE COLOR PALETTE SCREEN
*
       LI   R0,>100        INITIALIZE WORD COUNTER
       LI   R1,PALET       INITIALIZE POINTER FOR
                           PALET ARRAY
LOOP3  MOV  @GRAY,*R1+     STORE GRAY COLOR >EEEE
       DEC  R0             DEC WORD COUNTER
       JNE  LOOP3          WRITE NEXT WORD
       CLR  R0             INITIALIZE COLOR VALUE
       LI   R3,16          INITIALIZE COLOR COUNTER
LOOP4  LI   R4,2           INITIALIZE COLUMN COUNTER
LOOP5  MOVB @GRAY,*R1+     STORE GRAY BYTE
       MOVB @GRAY,*R1+     STORE ANOTHER GRAY BYTE
       MOVB @BLACK,*R1+    STORE BLACK BYTE
       LI   R5,4           LOAD COUNTER FOR COLOR BYTES
LOOP6  MOVB R0,*R1+        STORE A COLOR BYTE
       DEC  R5             DEC COLOR BYTE COUNTER
       JNE  LOOP6          STORE ANOTHER COLOR BYTE
       MOVB @BLACK,*R1+    STORE A BLACK BYTE
       DEC  R4             DEC COLUMN COUNTER
       JNE  LOOP5          DO SECOND COLUMN
       SWPB R0             SHIFT TO LEAST SIG BYTE
       AI   R0,>11         ADD 1 FOR NEXT COLOR NUMBER
       SWPB R0             SHIFT BACK TO MOST SIG BYTE
       DEC  R3             COUNT DOWN COLOR COUNTER
       JNE  LOOP4          DO NEXT TWO COLUMNS
       LI   R0,>300        SET BYTE COUNTER FOR
                           REMAINING SCREEN
```

Continued on p. 85

Dynamic

```
70 DATA LO,10,4,96,LI,10,5,
   136,LO,9,4,96,LI,9,5,136
71 DATA LO,5,6,96,CT,5,7,138,
   LO,6,6,96,LI,6,7,136,LO,7,
   5,96,LI,7,6,136,
   LO,8,5,96,LI,8,6,136
72 DATA RO,5,8,97,RO,6,8,97,
   RI,6,7,137,RO,7,8,97,RI,7,
   7,137,RO,8,9,97,
   RI,8,8,137,RO,9,10,97
73 DATA RI,9,9,137,RO,10,10,
   97,RI,10,9,137,RO,11,11,
   97,RI,11,10,137,
   RO,12,11,97, RI,12,10,137
74 DATA RO,13,11,97,RI,13,10,
   137,RO,14,12,97,RI,14,11,
   137,RO,15,12,97,
   RI,15,11,137,LO,16,12,97
75 DATA LI,16,11,137,LO,17,
   13,97,LI,17,12,137,LO,18,
   13,97,B,18,12,137,
   B,19,12,99,BI,19,11,139
76 DATA BI,19,10,140,
   BO,19,9,100
77 DATA BOTTOM,20,5,96,B,20,
   6,99,B,19,7,139,B,19,8,99,
   B,19,4,100,B,19,5,140,
   B,19,2,96,B,19,3,99
78 DATA B IN,18,7,139,
   B OUT,18,8,136
79 REM  ----CROSS----
80 DATA TOP,2,7,152,L RADIAL,
   2,6,153,R RADIAL,2,8,153,
   T RADIAL,1,7,154,B RADIAL,
   3,7,154,B RAD,4,7,154
81 REM  ----ORNAMENTS----
82 DATA OUTSIDE BELL,7,9,104,
   PLUM,20,2,107,PLUM,14,4,
   112,DIAMOND,13,9,113,
   BELL,16,10,115
83 DATA PLUM,13,4,112,
   PLUM,12,9,128,
   DIAMOND,14,4,129,
   BELL,17,3,131
84 REM SCREEN LOCATION LOOP
85 HOWMANY=86
86 RESTORE 66
87 FOR CHARACTER=1 TO HOWMANY
88 READ IDENTIFICATION$,ROW,
   COLUMN,CHARACTERNUMBER
89 CALL HCHAR(ROW,COLUMN,
   CHARACTERNUMBER)
90 NEXT CHARACTER
91 CALL KEY(0,K,S)
92 IF S=0 THEN 91
93 END
```

Listing 2

```
1 REM  **************
2 REM  * BAR-TOPPER *
3 REM  **************
4 REM  BY FRED ELLIS
5 REM   99'ER VERSION 1.6.1
6 REM  ABOUT 5392 BYTES
7 REM  PRESS ANY KEY TO
       STOP DISPLAY.'
8 VERTICALMAX=200
9 SCALE=VERTICALMAX/20
10 CALL CLEAR
11 LABEL$="ENTER HORSEPOWER"
12 ROW=12
13 COLUMN=15
14 GOSUB 91
15 LABEL$="0 TO 209"
16 ROW=13
17 COLUMN=19
18 GOSUB 91
19 INPUT "                ":
   HORSEPOWER
20 CALL SCREEN(8)
21 CALL COLOR(9,13,8)
22 CALL COLOR(10,2,5)
23 REM DEFINE CHARACTERS
24 REM  FORMAT:
   IDENTIFICATION$,
   CHARACTERNUMBER,
   HEXADECIMAL$...
25 REM  ---GRID---
26 DATA GRID LINE,91,
   00000000000000FF,
   VERTICAL AXIS,92,
   0101010101010101,
   TIC MARK,93,
   010101010101017F
27 REM  --DEFINE BAR TOPS--
28 DATA BOTTOM ROW OF PIXELS
   ON,96,00000000000000FF,
   SECOND ROW ON,97,
   000000000000FFFF,
   THIRD ROW ON
29 DATA 98,0000000000FFFFFF,
   FOURTH,99,
   00000000FFFFFFFF,FIFTH,
   100,000000FFFFFFFFFF,
   SIXTH,101,
   0000FFFFFFFFFFFF
30 DATA SEVENTH,102,
   00FFFFFFFFFFFFFF,
   EIGHTH,103,
   FFFFFFFFFFFFFFFF
31 REM  ---BASELINE---
32 DATA BASE,104,
   FF0000FF000000FF
33 REM DEFINE LOOP
34 RESTORE 26
35 FOR CODE=91 TO 104
36 READ IDENTIFICATION$,
   CHARACTERNUMBER,HEX$
37 IF CHARACTERNUMBER>CODE
   THEN 39
38 GOTO 40
39 CODE=CHARACTERNUMBER
40 CALL CHAR(CODE,HEX$)
41 NEXT CODE
42 REM START SCREEN DISPLAY
43 REM  ---GRAPH GRID---
44 CALL HCHAR(22,13,104,18)
45 FOR ROW=21 TO 1 STEP -1
46 CALL HCHAR(ROW,14,91,17)
47 NEXT ROW
48 LABEL$="HORSEPOWER"
49 ROW=9
50 COLUMN=1
51 GOSUB 91
52 CALL VCHAR(1,13,92,21)
53 FOR ROW=21 TO 1 STEP -5
54 ROWNUMBER=200-(10*(ROW-1))
55 LABEL$=STR$(ROWNUMBER)
56 COLUMN=10
57 GOSUB 91
58 CALL HCHAR(ROW,13,93)
59 NEXT ROW
60 REM CALCULATE BAR HEIGHT
61 BARHEIGHT=HORSEPOWER/SCALE
62 Y=INT(BARHEIGHT)
63 REMAINDER=BARHEIGHT-INT
   (BARHEIGHT)
64 CALL VCHAR(22-Y,16,103,Y)
65 CALL VCHAR(22-Y,17,103,Y)
66 CALL VCHAR(22-Y,18,103,Y)
67 REM SELECT BAR TOP
68 TOPPATTERN=INT
   ((REMAINDER*8)+.5)
69 ON TOPPATTERN+1 GOTO 70,
   72,74,76,78,80,82,84,86
70 CALL HCHAR(21-Y,16,96,3)
71 GOTO 88
72 CALL HCHAR(21-Y,16,97,3)
73 GOTO 88
74 CALL HCHAR(21-Y,16,98,3)
75 GOTO 88
76 CALL HCHAR(21-Y,16,99,3)
77 GOTO 88
78 CALL HCHAR(21-Y,16,100,3)
79 GOTO 88
80 CALL HCHAR(21-Y,16,101,3)
81 GOTO 88
82 CALL HCHAR(21-Y,16,102,3)
83 GOTO 88
84 CALL HCHAR(21-Y,16,103,3)
85 GOTO 88
86 CALL HCHAR(21-Y,16,103,3)
87 CALL HCHAR(20-Y,16,96,3)
88 CALL KEY(0,K,S)
89 IF S=0 THEN 88
90 END
91 FOR POSITION=1 TO
   LEN(LABEL$)
92 LETTER$=SEG$(LABEL$,
   POSITION,1)
93 CODE=ASC(LETTER$)
94 CALL HCHAR(ROW,
   COLUMN-1+POSITION,CODE)
95 NEXT POSITION
96 RETURN
```

Listing 3

```
1 REM  ************
2 REM  * AUTO-TOP *
3 REM  ************
4 REM  BY FRED ELLIS
5 REM   99'ER VERSION 1.6.1
6 REM  ABOUT 5288 BYTES
7 REM  PRESS ANY KEY TO STOP
       DISPLAY.
8 VERTICALMAX=200
9 SCALE=VERTICALMAX/20
10 CALL CLEAR
11 LABEL$="ENTER HORSEPOWER"
12 ROW=12
13 COLUMN=15
14 GOSUB 78
15 LABEL$="0 TO 209"
16 ROW=13
17 COLUMN=19
18 GOSUB 78
19 INPUT "               ":
   HORSEPOWER
20 CALL SCREEN(8)
21 CALL COLOR(9,13,8)
22 CALL COLOR(10,2,5)
23 REM DEFINE CHARACTERS
24 REM  FORMAT:
   IDENTIFICATION$,
   CHARACTERNUMBER,
   PATTERN$...
25 DATA GRID LINE,91,
   00000000000000FF,
   VERTICAL AXIS,92,
   0101010101010101,
   TIC MARK,93,
   010101010101017F
26 DATA BAR,96,
   FFFFFFFFFFFFFFFF,
   BASELINE,104,
   FF0000FF000000FF
27 DATA RESERVED FOR
   TITLE BOX
28 DATA RESERVED FOR LABELS
29 DATA RESERVED FOR LEGEND
30 DATA RESERVER FOR
   ADDITIONAL CHARACTERS
31 REM DEFINE-LOOP
32 RESTORE 25
33 FOR CODE=91 TO 104
34 READ IDENTIFICATION$,
   CHARACTERNUMBER,PATTERN$
35 IF CHARACTERNUMBER>CODE
   THEN 37
36 GOTO 38
37 CODE=CHARACTERNUMBER
38 CALL CHAR(CODE,PATTERN$)
39 NEXT CODE
40 REM START SCREEN DISPLAY
41 REM  ---GRAPH GRID---
42 CALL HCHAR(22,13,104,18)
43 FOR ROW=21 TO 1 STEP -1
44 CALL HCHAR(ROW,14,91,17)
45 NEXT ROW
46 LABEL$="HORSEPOWER"
47 ROW=9
48 COLUMN=1
49 GOSUB 78
50 CALL VCHAR(1,13,92,21)
51 FOR ROW=21 TO 1 STEP -5
52 ROWNUMBER=200-(10*(ROW-1))
53 LABEL$=STR$(ROWNUMBER)
54 COLUMN=10
55 GOSUB 78
56 CALL HCHAR(ROW,13,93)
57 NEXT ROW
58 REM CALCULATE BAR HEIGHT
59 BARHEIGHT=HORSEPOWER/SCALE
60 Y=INT(BARHEIGHT)
61 REMAINDER=BARHEIGHT-INT
   (BARHEIGHT)
62 CALL VCHAR(22-Y,16,96,Y)
63 CALL VCHAR(22-Y,17,96,Y)
64 CALL VCHAR(22-Y,18,96,Y)
65 REM  SELECT BAR TOP
66 TOPPATTERN=1+INT
   ((REMAINDER*8)+.5)
67 MASTER$="0000000000000
   FFFFFFFFFFFFFFFFF"
68 STARTPOSITION=2*
   TOPPATTERN-1
69 TOPPATTERN$=SEG$(MASTER$,
   STARTPOSITION,16)
70 CALL CHAR(97,TOPPATTERN$)
71 CALL HCHAR(21-Y,16,97,3)
72 IF TOPPATTERN<>9 THEN 75
73 CALL CHAR(98,
   "00000000000000FF")
74 CALL HCHAR(20-Y,16,98,3)
75 CALL KEY(0,K,S)
76 IF S=0 THEN 75
77 END
78 FOR POSITION=1 TO
   LEN(LABEL$)
79 LETTER$=SEG$
   (LABEL$,POSITION,1)
80 CODE=ASC(LETTER$)
81 CALL HCHAR(ROW,
   COLUMN-1+POSITION,CODE)
82 NEXT POSITION
83 RETURN
```


Listing 4

```
1 REM  **************
2 REM  * THREE-BARS *
3 REM  **************
4 REM  BY FRED ELLIS
5 REM   99'ER VERSION 1.6.1
6 REM  ABOUT 5160 BYTES
7 REM  PRESS ANY KEY TO STOP
       DISPLAY.
8 VERTICALMAX=200
9 SCALE=VERTICALMAX/20
10 OPTION BASE 1
11 DIM Y(3)
12 Y(1)=133
13 Y(2)=159
14 Y(3)=99.9
15 CALL SCREEN(8)
16 CALL COLOR(9,5,8)
17 CALL COLOR(10,3,8)
18 CALL COLOR(11,16,8)
19 CALL COLOR(12,2,5)
20 REM DEFINE CHARACTERS
21 REM  FORMAT:
   IDENTIFICATION$,
   CHARACTERNUMBER,
   PATTERN$...
22 DATA GRID LINE,91,
   00000000000000FF,
   VERTICAL AXIS,92,
   0101010101010101,
   TIC MARK,93,
   010101010101017F,
   BAR1,96
23 DATA FFFFFFFFFFFFFFFF,
   BAR2,104,
   FFFFFFFFFFFFFFFF,
   BAR3,112,
   FFFFFFFFFFFFFFFF,
   BASELINE,120,
   FF0000FF000000FF
24 DATA RESERVED FOR
   TITLE BOX
25 DATA RESERVED FOR LABELS
26 DATA RESERVED FOR LEGEND
27 DATA RESERVER FOR
   ADDITIONAL CHARACTERS
28 REM DEFINE-LOOP
29 RESTORE 22
30 FOR CODE=91 TO 120
31 READ IDENTIFICATION$,
   CHARACTERNUMBER,PATTERN$
32 IF CHARACTERNUMBER>CODE
   THEN 34
33 GOTO 35
34 CODE=CHARACTERNUMBER
35 CALL CHAR(CODE,PATTERN$)
36 NEXT CODE
37 REM START SCREEN DISPLAY
38 CALL CLEAR
39 PRINT TAB(13);Y(1);
   TAB(18);Y(2);Y(3)
40 REM  ---GRAPH GRID---
41 CALL HCHAR(22,13,120,18)
42 FOR ROW=21 TO 1 STEP -1
43 CALL HCHAR(ROW,14,91,17)
44 NEXT ROW
45 LABEL$="HORSEPOWER"
46 ROW=9
47 COLUMN=1
48 GOSUB 78
49 CALL VCHAR(1,13,92,21)
50 FOR ROW=21 TO 1 STEP -5
51 ROWNUMBER=200-(10*(ROW-1))
52 LABEL$=STR$(ROWNUMBER)
53 COLUMN=10
54 GOSUB 78
55 CALL HCHAR(ROW,13,93)
56 NEXT ROW
57 REM CALCULATE & PLOT BARS
58 MASTER$="0000000000000
   FFFFFFFFFFFFFFFFF"
59 FOR BAR=1 TO 3
```

Continued on p. 86

Crayon . . . from p. 83

```
LOOP7  MOVB @GRAY,*R1+           STORE A GRAY BYTE
       DEC  R0                   COUNT DOWN
       JNE  LOOP7                REPEAT UNTIL DONE
*
* INITIALIZE PATTERN TABLE - TRANSPARENT
*
CLEAR  LI   R0,>300              INITIALIZE WORD COUNTER
       LI   R1,PATRN             INITIALIZE POINTER FOR PATTERN ARRAY
LOOP8  MOV  @ZERO,*R1+           STORE COLOR = TRANSPARENT
       DEC  R0                   COUNT DOWN FOR NEXT WORD
       JNE  LOOP8                WRITE NEXT WORD IN ARRAY
*
* LOAD PATTERN TABLE
*
       LI   R0,>800              VDP PATTERN TABLE ADDRESS
       LI   R1,PATRN             CPU BUFFER ADDRESS
       LI   R2,>600              1536 BYTES TO WRITE
       BLWP @VMBW                WRITE TO VDP RAM
*
* SELECT DOUBLE SIZE AND MULTICOLOR MODE
*
       LI   R0,>01EA             TO WRITE 11101010 TO VDP R1
       BLWP @VWTR                WRITE TO VDP R1
       SWPB R0                   MOVE >EA TO MOST SIG BYTE
       MOVB R0,@>83D4            STORE COPY (>EA) IN CPU RAM
*
* DEFINE ATTRIBUTES FOR SPRITE #0
*
       LI   R0,>300              VDP SPRITE ATTRIBUTE LIST
       LI   R1,ATTRIB            LOCATION OF ATTRIBUTE LIST
                                 FOR SPRITE 0
       LI   R2,6                 6 BYTES TO MOVE
       BLWP @VMBW                WRITE DATA TO VDP RAM
*
* DEFINE # OF ACTIVE SPRITES
*
       MOVB @ONE,@SPRITE         STORE NO. OF ACTIVE SPRITES
                                 IN CPU RAM
*
* INITIALIZE CURSOR COLOR AND COLOR CHANGE COUNTER
*
       LI   R3,>0F01             SPRITE COLORS - WHITE/BLACK IN R3
       CLR  R4                   INITIALIZE COUNTER - COLOR CHANGE
*
* ---------- START MAIN LOOP ----------------
*
* CHECK JOYST FOR MOTION, FIRE BUTTON AND KEYS
*
CHECK  LIMI 2                    ENABLE INTERRUPTS
       LIMI 0                    DISABLE INTERRUPTS
       LI   R0,1                 INDICATE REPETIONS OF CHECKS
       BL   @CHECKS              BRANCH TO SUBROUTINE CHECKS
       MOVB @ONE,@UNIT           SELECT REMOTE UNIT TO SCAN
       BLWP @KSCAN               SCAN LEFT KEYBOARD
       CB   @FIRE,@H05           WAS "E" PRESSED?
       JEQ  CLEAR                IF YES GO TO CLEAR SCREEN
       CB   @FIRE,@H02           WAS "S" PRESSED?
       JNE  NEXT1                IF NOT, GO ON
       B    @SAVE                IF SO, BRANCH TO SAVE ROUTINE
NEXT1  CB   @FIRE,@H06           WAS "R" PRESSED?
       JNE  NEXT2                IF NOT, GO ON
       B    @RECALL              IF SO, BRANCH TO RECALL ROUTINE
NEXT2  CB   @FIRE,@H11           WAS "T" PRESSED?
       JNE  NEXT3                IF NOT, GO ON
       LIMI 2                    ENABLE INTERRUPTS
       LWPI GPLWS                LOAD GPL WORK SPACE
       BLWP @0000                RETURN TO MASTER TITLE SCREEN
NEXT3  CB   @FIRE,@H14           WAS "C" PRESSED?
       JNE  NEXT4                IF NO, GO ON
       B    @SELECT              IF YES, GO TO COLOR SELECT ROUTINE
NEXT4  CB   @FIRE,@H18           WAS FIRE BUTTON PRESSED?
       JNE  SKIP                 IF NO, SKIP DRAW ROUTINE
*
* ROUTINE TO PLACE BLOCK ON SCREEN
*
DRAW   LI   R0,>300              VDP SPRITE ATTRIBUTE ADDRESS
       LI   R1,ROW               CPU BUFFER TO RECEIVE DATA
       LI   R2,2                 FETCH 2 BYTES
       BLWP @VMBR                FETCH DOT ROW AND DOT COLUMN
       CLR  R7                   INITIALIZE R7 AND R8
       CLR  R8                   --FOR USE IN DIVIDE OPERATION
       CLR  R2                   INITIALIZE OFFSET FOR PATRN ARRAY
       MOVB @ROW,R8              PUT DOT ROW IN R8
       SWPB R8                   MAKE IT LEAST SIG BYTE
       AI   R8,9                 ADD ROW OFFSET FOR COLOR BLOCK +1
       C    R8,@COLMAX           IS THE DOT ROW > 255?
       JLT  NOCORR               IF NOT, DO NOT APPLY CORRECTION
       S    @COLMAX,R8           IF SO, SUBTRACT 255
NOCORR DIV  @D32,R7              DIVIDE DOT ROW OF BLOCK BY 32
       SLA  R7,8                 CALCULATE BYTES IN PRECEEDING GROUPS
       A    R7,R2                ADD # OF BYTES IN PREVIOUS 32X8 BYTE
                                 GROUPS
       SRL  R8,2                 DIVIDE REMAINDER BY 4
       A    R8,R2                ADD # BYTES ABOVE IN CURRENT 8 BYTE
                                 SET
       CLR  R7                   INITIALIZE R7 AND R8
       CLR  R8                   --FOR USE IN DIVIDE OPERATION
       MOVB @COL,R8              PUT DOT COLUMN IN R8
       SWPB R8                   MAKE IT LEAST SIG BYTE
       AI   R8,8                 ADD COLUMN OFFSET FOR COLOR BLOCK
       C    R8,@COLMAX           IS THE DOT COLUMN > 255?
       JLT  NOCORC               IF NOT, DO NOT APPLY CORRECTION
       S    @COLMAX,R8           IF SO, SUBTRACT 256
NOCORC DIV  @D8,R7               DIVIDE BY 8
       SLA  R7,3                 CALCULATE BYTES IN PRECEEDING 8 BYTE
                                 SETS
       A    R7,R2                ADD # BYTES IN PREVIOUS 8 BYTE SETS,
                                 THIS GROUP
       MOV  R2,R2                CHECK IF INSIDE PATTERN ARRAY  N
       JLT  SKIP                 IF NOT SKIP SCREEN PLACEMENT
       C    R2,@MAX              CHECK IF INSIDE PATTERN ARRAY EEN
       JGT  SKIP                 IF NOT SKIP SCREEN PLACEMENT
       LI   R0,>14               REPEAT SUBROUTINE CHECKS 20 TIMES
       BL   @CHECKS              BRANCH TO SUBROUTINE CHECKS
       CLR  R1                   INITIALIZE R1 FOR BLOCK COLOR
       MOVB @FCOLOR,R1           STORE COLOR IN R1
       SWPB R1                   MAKE IT LEAST SIG BYTE
       CLR  R0                   INITIALIZE R0 FOR CURRENT ARRAY
                                 ELEMENT
       MOVB @PATRN(2),R0         COPY ARRAY ELEMENT AT OFFSET INTO R0
       SRL  R8,2                 CALCULATE WHETHER BLOCK IS LEFT OR
                                 RIGHT
       JEQ  MARK1                IF 0 LEAVE BLOCK AS LEFT NYBBLE
       SRL  R1,4                 IF 1 MAKE BLOCK RIGHT NYBBLE
       SWPB R0                   MAKE CURRENT ELEMENT LEAST SIG BYTE
       SRL  R0,4                 GET RID OF LEAST SIG NYBBLE
       SLA  R0,4                 PUT REMAINING NYBBLE BACK
       JMP  MARK2                SKIP TO LABEL
```

Continued on p. 94

Index to Advertisers

Crayon . . . from p. 85

```
MARK1  SLA  R0,4              GET RID OF MOST SIG NYBBLE
       SRL  R0,4              PUT BACK REMAINING NYBBLE
       SWPB R0                MAKE IT LEAST SIG BYTE
MARK2  A    R1,R0             ADD NEW COLOR TO ADJACENT VALUE
       SWPB R0                MAKE IT MOST SIG BYTE
       MOVB R0,@PATRN(2)      MOVE IT TO ARRAY AT OFFSET
       LI   R0,>0800          VDP PATTERN TABLE ADDRESS
       LI   R1,PATRN          CPU BUFFER
       LI   R2,>600           1536 BYTES TO MOVE
       BLWP @VMBW             WRITE TO REDRAW SCREEN
SKIP   CLR  R5                CLEAR R5 AND R6 TO RECEIVE JOYST VALUES
       CLR  R6
       MOVB @JOYSTY,R5        PUT Y RETURN IN R5
       NEG  R5                MULTIPLY BY -1
       SLA  R5,2              MULTIPLY BY 4
       MOVB @JOYSTX,R6        PUT X RETURN IN R6
       SLA  R6,2              MULTIPLY TIMES 4
       SWPB R6                MAKE XVEL LEAST SIG BYTE
       MOVB R5,R6             MOVE YVEL TO R6 AS MOST SIG BYTE
       LI   R1,USRWS+12       CPU ADDRESS OF VELOCITY BYTES (R6)
       LI   R0,>0780          VDP ADDRESS OF MOTION TABLE
       LI   R2,2              2 BYTES TO MOVE
       BLWP @VMBW             WRITE DATA TO VDP RAM
       B    @CHECK             START LOOP OVER AGAIN
*
* ----- END OF MAIN PROGRAM LOOP --------------------
*
* COLOR SELECT ROUTINE
*
SELECT LI   R0,>07EE          CHANGE BACKGROUND TO GRAY
       BLWP @VWTR             WRITE TO VDP R7
       LI   R0,>800           VDP BUFFER FOR PATTERN TABLE
       LI   R1,PALET          CPU BUFFER FOR PALETTE
       LI   R2,>600           1536 BYTES TO MOVE
       BLWP @VMBW             DISPLAY PALETTE
       LI   R0,>300           VDP BUFFER FOR ATTRIBUTE LIST
       LI   R1,ARRATT         ARROW ATTRIBUTES
       LI   R2,4              4 BYTES TO MOVE
       BLWP @VMBW             WRITE DATA
       BL   @DEBNC            BRANCH TO "DEBOUNCE" SUBROUTINE
LOOP9  LIMI 2                 ENABLE VDP INTERRUPT
       LIMI 0                 DISABLE INTERRUPT
       MOVB @ONE,@UNIT        IDENTIFY REMOTE UNIT TO SCAN
       BLWP @KSCAN            SCAN LEFT KBD AND REMOTE UNIT #1
       CB   @FIRE,@H18        CHECK FIRE BUTTON
       JEQ  CMARK             IF PRESSED, CHANGE MARK COLOR
       CB   @FIRE,@H14        CHECK "C" KEY
       JEQ  CSCRN             IF PRESSED, CHANGE SCREEN COLOR
       CLR  R6                INITIALIZE R6
       MOVB @JOYSTX,R6        PUT JOYST X IN R6
       SLA  R6,2              MPY BY 4
       SWPB R6                MAKE LEAST SIG BYTE
       LI   R1,USRWS+12       LOAD CPU ADDRESS (R6)
       LI   R0,>0780          LOAD ADDRESS OF MOTION TABLE
       LI   R2,2              MOVE 2 BYTES
       BLWP @VMBW             LOAD DATA TO VDP RAM
       JMP  LOOP9             GOTO LOOP9
CSCRN  BL   @DOTCOL           DETERMINE COLOR FROM DOT COLUMN OF ARROW
       SWPB R1                MAKE IT MOST SIG BYTE
       MOVB R1,@BCOLOR        MOVE IT TO BCOLOR
       JMP  BACK              JUMP TO BACK
CMARK  BL   @DOTCOL           DETERMINE COLOR FROM DOT COLUMN OF ARROW
       SLA  R1,12             PUT IN PROPER POSITION FOR @FCOLOR
       MOVB R1,@FCOLOR        MOVE IT TO FCOLOR
BACK   BL   @DEBNC            DEBOUNCE
       CLR  R0                PREPARE TO RETURN SCREEN COLOR
       MOVB @BCOLOR,R0        PUT BACKGROUND COLOR IN R0
       SWPB R0                MAKE IT LEAST SIG BYTE
       MOVB @H07,R0           INDICATE WRITE TO VDP R7
       BLWP @VWTR             WRITE IT TO R7
       LI   R0,>800           VDP PATTERN TABLE ADDRESS
       LI   R1,PATRN          PATTERN BUFFER IN CPU RAM
       LI   R2,>600           1536 BYTES TO WRITE
       BLWP @VMBW             LOAD PATTERN SCREEN
       LI   R0,>300           VDP SPRITE ATTRIBUTE TABLE ADDRESS
       LI   R1,ATTRIB         ADDRESS OF CURSOR ATTRIBUTES
       LI   R2,4              4 BYTES TO MOVE
       BLWP @VMBW             LOAD DATA TO GET CURSOR SPRITE
       B    @SKIP             BRANCH TO LABLE SKIP
*
* DSR ROUTINE TO SAVE "SCREEN" -- PATTERN TABLE
*
SAVE   LI   R0,>1000          PREPARE TO MOVE PATRN TO VDP BUFFER
       LI   R1,PATRN          CPU BUFFER ADDRESS
       LI   R2,>600           1536 BYTES TO MOVE
       BLWP @VMBW             WRITE DATA
       LI   R0,PAB            VDP PERIPHERAL ACCESS BLOCK ADDRESS
       LI   R1,PDATA          CPU BUFFER TO BE WRITTEN TO VDP
       LI   R2,21             21 BYTES TO WRITE
       BLWP @VMBW             WRITE PAB
       LI   R6,PAB+9          SET POINTER TO NAME LENGTH
       MOV  R6,@PNTR          STORE IN >8356 >8357
       BLWP @DSRLNK           EXECUTE SAVE OR LOAD
       DATA 8
       B    @CHECK            IF SO, BRANCH BACK TO BEGINNING
*
* DSR ROUTINE TO RECALL "SCREEN" -- PATTERN TABLE
*
RECALL LI   R0,PAB            VDP PERIPHERAL ACCESS BLOCK ADDRESS
       LI   R1,PDATA          CPU BUFFER TO WRITE
       LI   R2,21             21 BYTES TO WRITE
       BLWP @VMBW             WRITE PAB
       LI   R0,PAB            SUBSTITUTE "LOAD" I/O OP CODE
       MOVB @LOAD,R1          MOVE OP CODE TO R1
       BLWP @VSBW             WRITE BYTE TO PAB
       LI   R6,PAB+9          SET POINTER TO NAME LENGTH
       MOV  R6,@PNTR          STORE IN >8356 >8357
       BLWP @DSRLNK           COPY DATA TO VDP BUFFER
       DATA 8
       LI   R0,>1000          PREPARE TO COPY FROM VDP TO PATRN
       LI   R1,PATRN          CPU BUFFER ADDRESS
       LI   R2,>600           1536 BYTES TO COPY
       BLWP @VMBR             COPY BUFFER
       LI   R0,>0800          NOW COPY TO PATTERN TABLE
       LI   R1,PATRN          ADDRESS OF CPU BUFFER
       LI   R2,>600           1536 BYTES TO COPY
       BLWP @VMBW             COPY TO TABLE
       B    @CHECK            BACK TO THE BEGINNING
*
* SUBROUTINE TO PERIODICALLY CHANGE SPRITE COLORS
*
CHECKS AI   R4,>100           ADD 256 TO R4
       JEQ  CHANGE            WHEN R4 REACHES 0, CHANGE COLOR
       DEC  R0                DEC COUNTER
       JNE  CHECKS            IF NOT 0 ADD ANOTHER 256
       JMP  RETURN            BACK TO MAIN PROGRAM
CHANGE SWPB R3                SWITCH COLOR BYTES IN R3
       MOV  R3,R1             PUT R3 IN R1
       LI   R0,>303           ADDRESS OF SPRITE #0 COLOR IN VDP RAM
       BLWP @VSBW             WRITE MOST SIG BYTE OF R1
RETURN RT                     BACK TO MAIN PROGRAM
*
* DEBOUNCE SUBROUTINE
*
DEBNC  MOVB @ONE,@UNIT        KEY UNIT TO CHECK
       BLWP @KSCAN            SCAN KEYBOARD
       CB   @FIRE,@NOKEY      IS NO KEY PRESSED?
       JNE  DEBNC             IF A KEY IS PRESSED, CHECK AGAIN.
       RT                     GO BACK TO MAIN PROGRAM
*
* SUBROUTINE TO DETERMINE COLOR FOR ARROW
*
DOTCOL CLR  R1                INITIALIZE R1 TO RECEIVE DOT COLUMN
       LI   R0,>301           VDP ADDRESS OF DOT COLUMN
       BLWP @VSBR             READ BYTE FROM ATTRIBUTE TABLE
       SWPB R1                MAKE IT LEAST SIG BYTE
       AI   R1,>07            ADD OFFSET FOR POINT OF ARROW
       SRL  R1,4              DIVIDE BY 16
       RT                     RETURN
*
* "END START"
*
AUTO   END  MARKER            AUTOSTART
```

DeBUGS on DisPlay

99'er Program Bug

LISTING #1 (SEGMENTED)
SCREEN DUMP PROGRAM FOR MINI-MEMORY
BY PATRICIA SWIFT 99'ER VERSION 2.1.2

```
ADDR  LABEL  OPCODE  OPERANDS      COMMENTS
----  -----  ------  --------      --------
7D14         MOVB    @>9802,@S1    GET MSB OF GROM ADDR INTO S1
             SWPB    @S1
             MOVB    @>9802,@S1    GET LSB OF GROM ADDR
             SWPB    @S1
             DEC     @S1           CORRECT FOR AUTO-INCREMENT
             LI      0,>1D00
             LI      1,PD
             LI      2,36
             BLWP    @>6028        WRITE PAB TO VDP RAM
             LI      6,>1D09
             MOV     6,@>8356      POINT TO DEVICE NAME LENGTH
             BLWP    @>6038        DSRLNK TO OPEN PRINTER
             DATA    8
             LI      10,>0400        <<** changed instruction **>>
             MOV     10,@>7DEA       <<** changed instruction **>>
                                     <<** deleted instruction **>>
7D52         LI      0,>1D00

 .            .        .
 .            .        .
 .            .        .

7DD2  L1     INC     3             POINT TO NEXT INPUT BYTE
             SRA     6,1           /2
             JGT     L2            DO NEXT BYTE, IF MORE
             SWPB    4             PUT OUTPUT BYTE IN MSB OF R4
             MOVB    4,@DO(8)      STORE AT DO
             INC     8             POINT TO NEXT BYTE OF DO
             SRA     5,1           /2
             JGT     L3            CONSTRUCT NEXT OUTPUT BYTE
             LI      0,>1D05
             LI      1,>0000         <<** changed instruction **>>
             BLWP    @>6024        PUT LENGTH OF 4 IN PAB
             LI      0,>1E00
             LI      1,E1
             LI      2,4
             BLWP    @>6028        PUT ESC K SEQ. IN DATA BUFF
             LI      6,>1D09
             MOV     6,@>8356      POINT TO DEVICE NAME LENGTH
             BLWP    @>6038        DSRLNK TO WRITE ESC K SEQ.
             DATA    8
             LI      10,>0000        <<** changed instruction **>>
             MOV     10, @>7DEA      <<** changed instruction **>>
                                     <<** deleted instruction **>>
             LI      0,>1D05
             LI      1,>0800

 .            .        .
 .            .        .
 .            .        .

7E78         LI      10,>0400        <<** changed instruction **>>
             MOV     10,@>7DEA       <<** changed instruction **>>
                                     <<** deleted instruction **>>
             JMP     L0            DO NEXT SCREEN CHARACTER
7E82  L4     LI      0,>1D00       COME HERE WHEN FINISHED DUMP
 .            .        .
 .            .        .
 .            .        .

7ED2  PD     DATA    >0012,>1E00,>FF00,>0000,>001A
*                                  PAB DEFINITION
             TEXT    'RS232.PA=O.DA=8.BA=9600.CR'<<*change*>>
*                                  DEVICE NAME
7EF6  CR     DATA    >0D0A         CR LF
7EF8  E1     DATA    >1B4B,>FF00       <<** changed line **>>
*                                  ESC K GRAPHICS SEQUENCE
7EFC  S1     BSS     2             SAVE AREA
7EFE  E2     DATA    >0D1B,>4108
*                                  CR AND ESC A VERT SPACING
7F02         END
```

A Screen Dump Utility

After working with **Super Language: A Screen Dump Utility—Part 2** in the November 1982 issue, we came up with an improvement that we just had to pass along. The changes shown here, in the segments of **Listing #1** from that November article, cause the screen dump program to write one full screen line at a time to the printer. This really speeds up the dump and reduces the printer-head motion.

The original DEBUGs for Screen Dump which appeared in the December 1982 issue on page 40 should be ignored. The File Descriptor *should* be:
RS232.PA = O.DA = 8.BA = 9600
That part of the Screen Dump corrections having to do with an error on page 24 of the November issue (column 2 under **Mini-Memory Considerations** Step 5) should read as follows:

5. Put the entry point for DUMP into the DEF/REF table by entering the following lines:

```
AORG >7FE8(CR)
TEXT 'DUMP  '(CR)
DATA >7D14(CR)
```

Note: There are 2 spaces required following the word DUMP in the above TEXT directive for a total of 6 characters within the apostrophes.

Tex-Scribe

A DEBUG in the **Tex-Scribe** article occured on page 16 of the December 1982 issue. Regarding **Table 2 "TI-99/4 Impact Printer Mode Commands,"** it was brought to our attention that the TI printer does not have an *Italics* mode. . . We designed that table using an *Epson MX-80 with Graftrax option* and mislabeled it. . . If you have the TI-99/4 Impact Printer, refer to your owner's manual for the available mode commands.

SUPER LANGUAGE
A Home Computer Assembly Language Series

MINI MEMORY DISASSEMBLER UTILITY

A Program By Martin Kroll, Jr.
218 Kaplan Ave.
Pittsburgh, PA 15227

Have you ever tried entering an Assembly Language program line by line into the Mini Memory cartridge only to find it doesn't work? Somewhere you typed in a wrong operation code . . . and you find yourself re-entering the entire program because you have no idea where the error is. That is the reason we on the 99'er staff were so pleased to see a program submitted for publication that can, if used properly, ease most of the pain: a disassembler written in TI BASIC that runs with the Mini Memory plugged in. And if you have an RS232 interface and a printer, you will be able to produce a hard-copy listing (or screen listing, without a printer) of the original Assembly Language. This *source* listing can then be studied to locate the error(s).

Here's the way it works: Once an Assembly Language program is "assembled" into machine code (the binary patterns on which the computer makes its decisions), it becomes very difficult for humans to read. Therefore, when debugging, it is a great help to convert this machine code into something we can understand. The disassembler program does just this by translating the machine code ("object") back into Assembly Language ("source") mnemonic statements. For example:

MACHINE CODE	ASSEMBLY LANGUAGE
>04C0	CLR R0

The ">" in the machine code simply means the value following it is hexadecimal (base 16). The Assembly Language mnemonic statement makes much more sense: "CLR R0" means CLeaR Register zero. The disassembler reads the value >04C0, determines the type of mnemonic code it is represented by, and prints the Assembly Language statement on the printer or screen.

The Program

Make sure your Mini Memory cartridge is loaded with the software you wish to disassemble, and that the cartridge is properly installed in the TI-99/4A. After loading the disassembler program under TI BASIC, type RUN. The message "WANT A PRINT OUT? Y/N" is displayed. Press Y and ENTER if you have a printer (N, if you want to display the disassembled code on the screen). The next message "DEVICE NAME?" is displayed if you chose Y. Enter the parameters for your printer. (For example, RS232.BA=9600.DA=8.) Once this is done, the master option screen appears:

1. DISASSEMBLE OPCODE
2. DISASSEMBLE DATA
3. DISASSEMBLE TEXT
4. FINISH

If option 1, 2, or 3 is selected, the message "DISASSEMBLE FROM? (4 DIGIT HEX ADDRESS)" is displayed. Enter the starting location in Mini Memory for the segment of machine code you wish to disassemble. The next prompt is "TO? (4 DIGIT HEX ADDRESS)." Enter the last address of the machine code segment. Mini Memory programs may reside anywhere between addresses >7000, and >7FFF. When actually entering the first and last address of the block of memory you wish to disassemble, you do not need to enter the "greater than" sign (>).

If option #1 is selected, the machine code will be interpreted as operation code instructions to the computer. In doing this, whenever the disassembler comes across data or text, either a "pseudo" mnemonic statement will be produced or the message "ILLEGAL OBJECT CODE" will be printed. After running option #1, you can get a good idea of where the data and text is located. Now you can use option #2, or #3.

Option #2 will print all machine code between the start and stop addresses as DATA statements. You will have to coordinate this print-out with the one you generated in option #1 if you do not know where the data is.

Option #3 will print all machine code from the starting to the stopping address in TEXT format. This means that all machine code will be treated as "ASCII" characters. If you try to print machine code in TEXT format which is not *printable text*, a question mark will be output for each non-printable character. Once you are finished disassembling, select option #4 to exit the program.

How It Works

With the Mini Memory installed, you have several new commands at your disposal in TI BASIC. One command which made this program possible is "CALL PEEK". It will return the decimal value of any memory location. Once it has the decimal value of a memory location, the program then converts that value to hexadecimal (base 16), and binary (base 2). The hexadecimal value is used in the printed report. The binary value is used to extract the control fields and operation code to ascertain the format and type of instruction that represents the machine code.

Some final notes: This disassembler cannot reconstruct the "labels" that you have used to mark portions of the program for branch or jump destinations. If you have the TI Memory Expansion, you will also find it possible to disassemble machine code in it with this disassembler utility. All in all, it is a very useful tool.

EXPLANATION OF THE PROGRAM
Mini Memory Disassembler

Line Nos.	
200-280	Initialize array, and set up printer.
290-410	Display main title screen and branch to options.
420-440	Subroutine to wait for Enter to be pressed.
450-580	Input start and stop addresses to be disassembled.
590-660	Get hexadecimal value of addresses.
670-820	Control loop to get a value from memory and convert it back to hexadecimal code.
830-920	Branch to formatting subroutines, depending on the code values.
930-1210	Subroutine to print disassembled listing.
1220-2950	Subroutines for instruction formatting.
1220-1380	Format #1.
1390-1620	Format #2.
1630-1750	Format #3.
1760-1890	Format #4.
1900-2020	Format #5.
2030-2410	Format #6.
2420-2470	Format #7.

Disassembler ..

Lines	Description
2480-2780	Format #8.
2790-2950	Format #9.
2960-3120	Convert binary to decimal.
3130-3200	Convert decimal to binary.
3210-3270	Get binary divisor.
3280-3370	Get the "T" field.
3380-3530	Set up operand fields.
3540-3570	Get the mnemonic of the op-code.
3580-3700	Convert decimal to hexadecimal.
3710-3900	Control loop for displaying DATA.
3910-4070	Control loop for displaying TEXT.
4080-4270	Display DATA on the screen.
4280-4440	Display the TEXT on the screen.
4450-4520	Subroutine to "PEEK" at a memory location.
4530	End of program.

```
100 REM  ***************
110 REM  * MINI-MEMORY *
120 REM  * DISASEMBLER *
130 REM  ***************
140 REM  BY MARTIN KROLL
150 REM  99'ER VERSION 2.5.1
160 REM
170 REM
180 REM
190 REM
200 DIM S(15)
210 GOSUB 3210
220 CALL CLEAR
230 INPUT "WANT A PRINTOUT?
      Y/N":PRINT$
240 IF PRINT$<>"Y" THEN 290
250 F=1
260 PRINT
270 INPUT "DEVICE NAME?
          ":DEVICE$
280 OPEN #1:DEVICE$
290 CALL CLEAR
300 PRINT "PRESS 1 - DISASSEMBLE O
    PCODE":"PRESS 2 - DISASSEMBLE
    DATA":"PRESS 3 - DISASSEMBLE T
    EXT":"PRESS 4 - FINISH"
310 CALL KEY(0,K,ST)
320 IF ST=0 THEN 310
330 IF (K<49)+(K>52)=-1 THEN 310
340 CALL CLEAR
350 IF K=52 THEN 4530
360 GOSUB 450
370 IF K=49 THEN 670
380 IF (K=50)+(F=1)=-2 THEN 3710
390 IF (K=50)+(F=0)=-2 THEN 4080
400 IF (K=51)+(F=1)=-2 THEN 3910
410 IF (K=51)+(F=0)=-2 THEN 4280 E
    LSE 4530
420 PRINT ::
430 INPUT "PRESS ENTER TO CONTINUE
    :":CON$
440 GOTO 290
450 REM   INPUT PROGRAM ADDRESS TO
     DISASSEMBLE
460 CALL CLEAR
470 INPUT "DIS-ASSEMBLE FROM ?
         (4 DIGIT HEX ADDRESS)
          ":A$
480 IF POS("13579BDF",SEG$(A$,LEN(
    A$),1),1)=0 THEN 500
490 A$=SEG$(A$,1,LEN(A$)-1)&SEG$("
    02468ACE",POS("13579BDF",SEG$(
    A$,LEN(A$),1),1),1)
500 IF LEN(A$)=4 THEN 530
510 PRINT ::"INPUT MUST HAVE 4 HEX
     DIGITS"::
520 GOTO 470
530 INPUT "TO ? (4 DIGIT HEX ADDRE
    SS)      ":B$
540 IF POS("13579BDF",SEG$(B$,LEN(
    B$),1),1)=0 THEN 560
550 B$=SEG$(B$,1,LEN(B$)-1)&SEG$("
    02468ACE",POS("13579BDF",SEG$(
    B$,LEN(B$),1),1),1)
560 IF LEN(B$)=4 THEN 590
570 PRINT ::"INPUT MUST HAVE 4 HEX
     DIGITS"::
580 GOTO 530
590 TEMP$=A$
600 GOSUB 2960
610 A=DEC
620 TEMP$=B$
630 GOSUB 2960
640 B=DEC
650 CALL CLEAR
660 RETURN
670 REM  PEEK VALUES & CONVERT
680 FOR LOC=A TO B STEP 2
690 L=0
700 V1=LOC
710 GOSUB 3580
720 LOC$=HEX$
730 GOSUB 4470
740 M=MX
750 N=NX
760 V=M*256+N
770 V1=V
780 GOSUB 3580
790 V$=HEX$
800 VA=V
810 GOSUB 3130
820 REM   DETERMINE INSTRUCTION FO
    RMAT
830 IF V<512 THEN 3690
840 IF V<832 THEN 2480
850 IF V<1024 THEN 2420
860 IF V<2048 THEN 2030
870 IF V<4096 THEN 1900
880 IF V<8192 THEN 1390
890 IF V<11264 THEN 1630
900 IF V<12288 THEN 2790
910 IF V<14336 THEN 1760
920 IF V<16384 THEN 2790 ELSE 1220
930 REM  PRINT MNEMONIC OP CODE
940 GAP=POS(OPER$," ",1)
950 IF GAP<6 THEN 980
960 E$=OPER$
970 GOTO 990
980 E$=SEG$(OPER$,1,GAP)&SEG$("
        ",1,5-GAP)&SEG$(OPER$,GAP+1
    ,LEN(OPER$))
990 IF F=0 THEN 1020
1000 PRINT #F:LOC$;"    ";V$;"    ";
     E$
1010 GOTO 1030
1020 PRINT #F:LOC$;" ";V$;" ";E$
1030 IF L=3 THEN 1060
1040 IF L=2 THEN 1160
1050 IF L<>1 THEN 1200
1060 IF F=0 THEN 1090
1070 PRINT #F:LO$(1);"    ";VV$(1)
1080 GOTO 1100
1090 PRINT #F:LO$(1);" ";VV$(1)
1100 IF L=1 THEN 1200
1110 IF F=0 THEN 1140
1120 PRINT #F:LO$(3);"    ";VV$(3);"
         ";OPE$
1130 GOTO 1200
1140 PRINT #F:LO$(3);" ";VV$(3);" "
     ;OPE$
1150 GOTO 1200
1160 IF F=0 THEN 1190
1170 PRINT #F:LO$(1);"    ";VV$(1):L
     O$(2);"    ";VV$(2)
1180 GOTO 1200
1190 PRINT #F:LO$(1);" ";VV$(1):LO$
     (2);" ";VV$(2)
1200 NEXT LOC
1210 GOTO 420
1220 REM  FORMAT I
1230 RESTORE 1370
1240 GOSUB 3540
1250 T$=SEG$(BIN$,11,2)
1260 R$=SEG$(BIN$,13,4)
1270 GOSUB 3070
1280 GOSUB 3280
1290 S$=R$
1300 T$=SEG$(BIN$,5,2)
1310 R$=SEG$(BIN$,7,4)
1320 GOSUB 3070
1330 GOSUB 3280
1340 D$=R$
1350 OPER$=OP$&" "&S$&","&D$
1360 GOTO 940
1370 DATA 61440,SOCB,57344,SOC,5324
     8,MOVB,49152,MOV,45056,AB,4096
     0,A
1380 DATA 36864,CB,32768,C,28672,SB
     ,24576,S,20480,SZCB,16384,SZC
1390 REM  FORMAT II
1400 RESTORE 1610
1410 GOSUB 3540
1420 DISP$=SEG$(BIN$,9,8)
1430 DIS=0
1440 FOR X=8 TO 15
1450 DIS=DIS+VAL(SEG$(DISP$,X-7,1))
     *S(X)
1460 NEXT X
1470 IF DIS<128 THEN 1490
1480 DIS=DIS-256
1490 IF SEG$(OP$,2,1)="B" THEN 1580
1500 IF DIS=0 THEN 1560
1510 DIS=DIS*2+LOC+2
1520 V1=DIS
1530 GOSUB 3580
1540 OPER$=OP$&" >"&HEX$
1550 GOTO 940
1560 OPER$="NOP"
1570 GOTO 940
1580 REM   CONTROL INSTRUCTION
1590 OPER$=OP$&" "&STR$(DIS)
1600 GOTO 940
1610 DATA 7936,TB,7680,SBZ,7424,SBO
     ,7168,JOP,6912,JH,6656,JL,6400
     ,JNO,6144,JOC,5888,JNC,5632,JN
     E,5376,JGT
1620 DATA 5120,JHE,4864,JEQ,4608,JL
     E,4352,JLT,4096,JMP
1630 REM  FORMAT III
1640 RESTORE 1750
1650 GOSUB 3540
1660 T$=SEG$(BIN$,11,2)
1670 R$=SEG$(BIN$,13,4)
1680 GOSUB 3070
1690 GOSUB 3280
1700 S$=R$
1710 R$=SEG$(BIN$,7,4)
1720 GOSUB 3070
1730 OPER$=OP$&" "&S$&","&STR$(R)
1740 GOTO 940
1750 DATA 10240,XOR,9216,CZC,8192,C
     OC
1760 REM  FORMAT IV
1770 RESTORE 1890
1780 GOSUB 3540
1790 R$=SEG$(BIN$,7,4)
1800 GOSUB 3070
1810 C$=STR$(R)
1820 R$=SEG$(BIN$,13,4)
1830 T$=SEG$(BIN$,11,2)
1840 GOSUB 3070
1850 GOSUB 3280
1860 S$=R$
1870 OPER$=OP$&" "&S$&","&C$
1880 GOTO 940
1890 DATA 13312,STCR,12288,LDCR
1900 REM  FORMAT V
1910 RESTORE 2020
1920 GOSUB 3540
1930 R$=SEG$(BIN$,13,4)
1940 GOSUB 3070
1950 S$="R"&STR$(R)
1960 C$=SEG$(BIN$,9,4)
1970 R$=C$
1980 GOSUB 3070
1990 D$=STR$(R)
2000 OPER$=OP$&" "&S$&","&D$
2010 GOTO 940
2020 DATA 2816,SRC,2560,SLA,2304,SR
     L,2048,SRA
2030 REM  FORMAT VI
2040 RESTORE 2150
2050 GOSUB 3540
2060 R$=SEG$(BIN$,13,4)
2070 T$=SEG$(BIN$,11,2)
2080 GOSUB 3070
2090 GOSUB 3280
2100 IF R$<"@>6018" THEN 2130
2110 IF R$>"@>6050" THEN 2130
2120 GOSUB 2170
2130 OPER$=OP$&" "&R$
2140 GOTO 940
2150 DATA 1856,ABS,1792,SETO,1728,S
     WPB,1664,BL,1600,DECT,1536,DEC
     ,1472,INCT,1408,INC
2160 DATA 1344,INV,1280,NEG,1216,CL
     R,1152,X,1088,B,1024,BLWP
2170 REM  MINI MEMORY UTILITY PROGR
     AMS
2180 DATA 6018,GPLLNK,601C,XMLLNK,6
     020,KSCAN,6024,VSBW,6028,VMBW,
     602C,VSBR,6030,VMBR
2190 DATA 6034,VWTR,6038,DSRLNK,603
     C,LOADER,6040,NUMASG,6044,NUMR
     EF,6048,STRASG,604C,STRREF,605
     0,ERR
2200 RESTORE 2180
2210 READ U$,UTIL$
2220 IF SEG$(R$,3,4)<>U$ THEN 2210
2230 IF F=0 THEN 2250
2240 R$=R$&"  ("&UTIL$&")"
2250 IF U$="6018" THEN 2290
2260 IF U$="6038" THEN 2290
2270 IF U$="601C" THEN 2290
2280 RETURN
2290 L=3
2300 LOC=LOC+2
2310 V1=LOC
2320 GOSUB 3580
2330 LO$(3)=HEX$
```

```
2340 GOSUB 4470
2350 M2=MX
2360 N2=NX
2370 V1=256*M2+N2
2380 GOSUB 3580
2390 VV$(3)=HEX$
2400 OPE$="DATA >"&HEX$
2410 RETURN
2420 REM  FORMAT VII
2430 RESTORE 2470
2440 GOSUB 3540
2450 OPER$=OP$
2460 GOTO 940
2470 DATA 992,LREX,969,SKOF,928,SKO
     N,896,RTWP,864,RSET,832,IDLE
2480 REM  FORMAT VIII
2490 RESTORE 2780
2500 GOSUB 3540
2510 R$=SEG$(BIN$,13,4)
2520 GOSUB 3070
2530 D$="R"&STR$(R)
2540 LOC=LOC+2
2550 L=L+1
2560 V1=LOC
2570 GOSUB 3580
2580 LO$(L)=HEX$
2590 GOSUB 4470
2600 M1=MX
2610 N1=NX
2620 V1=256*M1+N1
2630 GOSUB 3580
2640 VV$(L)=HEX$
2650 S$=">"&HEX$
2660 IF OP$="LIMI" THEN 2720
2670 IF OP$="LWPI" THEN 2720
2680 IF OP$="STST" THEN 2740
2690 IF OP$="STWP" THEN 2740
2700 OPER$=OP$&" "&D$&","&S$
2710 GOTO 940
2720 OPER$=OP$&" "&S$
2730 GOTO 940
2740 LOC=LOC-2
2750 L=L-1
2760 OPER$=OP$&" "&D$
2770 GOTO 940
2780 DATA 768,LIMI,736,LWPI,704,STS
     T,672,STWP,640,CI,608,ORI,576,
     ANDI,544,AI,512,LI
2790 REM  FORMAT IX
2800 RESTORE 2950
2810 GOSUB 3540
2820 R$=SEG$(BIN$,13,4)
2830 T$=SEG$(BIN$,11,2)
2840 GOSUB 3070
2850 GOSUB 3280
2860 S$=R$
2870 R$=SEG$(BIN$,7,4)
2880 GOSUB 3070
2890 IF OP$<>"XOP" THEN 2920
2900 D$=STR$(R)
2910 GOTO 2930
2920 D$="R"&STR$(R)
2930 OPER$=OP$&" "&S$&","&D$
2940 GOTO 940
2950 DATA 15360,DIV,14336,MPY,11264
     ,XOP
2960 REM  CONVERT TO DECIMAL
2970 DEC=0
2980 FOR X=3 TO 15 STEP 4
2990 TEMP2$=SEG$(TEMP$,(X+1)/4,1)
3000 IF ASC(TEMP2$)>57 THEN 3050
3010 TN=ASC(TEMP2$)-48
3020 DEC=DEC+TN*S(X)
3030 NEXT X
3040 RETURN
3050 TN=ASC(TEMP2$)-55
3060 GOTO 3020
3070 REM  GET REGISTER #
3080 R=0
3090 FOR X=12 TO 15
3100 R=R+VAL(SEG$(R$,X-11,1))*S(X)
3110 NEXT X
3120 RETURN
3130 REM  CONVERT TO BINARY
3140 BIN$=""
3150 FOR X=0 TO 15
3160 BIN=INT(VA/S(X))
3170 VA=VA-(BIN*S(X))
3180 BIN$=BIN$&STR$(BIN)
3190 NEXT X
3200 RETURN
3210 REM  GET BINARY DIVISOR
3220 DATA 32768,16384,8192,4096,204
     8,1024,512,256,128,64,32,16,8,
     4,2,1
3230 RESTORE 3220
3240 FOR X=0 TO 15
3250 READ S(X)
3260 NEXT X
3270 RETURN
3280 REM  GET T-FIELD
3290 IF T$<>"00" THEN 3320
3300 R$="R"&STR$(R)
3310 RETURN
3320 IF T$<>"01" THEN 3350
3330 R$="*R"&STR$(R)
3340 RETURN
3350 IF T$<>"11" THEN 3380
3360 R$="*R"&STR$(R)&"+"
3370 RETURN
3380 LOC=LOC+2
3390 L=L+1
3400 GOSUB 4470
3410 M1=MX
3420 N1=NX
3430 V1=LOC
3440 GOSUB 3580
3450 LO$(L)=HEX$
3460 V1=M1*256+N1
3470 GOSUB 3580
3480 VV$(L)=HEX$
3490 IF R<>0 THEN 3520
3500 R$="@>"&HEX$
3510 RETURN
3520 R$="@>"&HEX$&"(R"&STR$(R)&")"
3530 RETURN
3540 REM  GET MNEMONIC OF OP CODE
3550 READ OPV,OP$
3560 IF V<OPV THEN 3550
3570 RETURN
3580 REM  CONVERT TO HEX
3590 HEX$=""
3600 FOR X=3 TO 15 STEP 4
3610 VH=INT(V1/S(X))
3620 V1=V1-VH*S(X)
3630 IF VH>9 THEN 3670
3640 HEX$=HEX$&STR$(VH)
3650 NEXT X
3660 RETURN
3670 HEX$=HEX$&CHR$(VH+55)
3680 GOTO 3650
3690 OPER$="ILLEGAL OBJECT CODE"
3700 GOTO 940
3710 REM  DISPLAY DATA
3720 FOR LOOP=A TO B STEP 18
3730 V1=LOOP
3740 GOSUB 3580
3750 L$=HEX$
3760 PRINT #F:L$;"           ";"DAT
     A ";
3770 FOR LOC=LOOP TO LOOP+16 STEP 2
3780 GOSUB 4470
3790 M=MX
3800 N=NX
3810 V1=256*M+N
3820 GOSUB 3580
3830 IF LOC=LOOP+16 THEN 3860
3840 IF LOC>=B-1 THEN 3890
3850 PRINT #F:">";HEX$;",";
3860 NEXT LOC
3870 PRINT #F:">";HEX$
3880 NEXT LOOP
3890 PRINT #F:">";HEX$
3900 GOTO 420
3910 REM  DISPLAY TEXT
3920 FOR LOOP=A TO B STEP 54
3930 V1=LOOP
3940 GOSUB 3580
3950 PRINT #F:HEX$;"              ";"T
     EXT '";
3960 FOR LOC=LOOP TO LOOP+53
3970 GOSUB 4470
3980 M=MX
3990 IF (M<127)+(M>31)=-2 THEN 4010
4000 M=63
4010 PRINT #F:CHR$(M);
4020 IF LOC=B THEN 4060
4030 NEXT LOC
4040 PRINT #F:"'"
4050 NEXT LOOP
4060 PRINT #F:"'"
4070 GOTO 420
4080 REM  DISPLAY DATA ON SCREEN
4090 FOR LOOP=A TO B STEP 6
4100 V1=LOOP
4110 GOSUB 3580
4120 L$=HEX$
4130 PRINT #F:L$;" DATA ";
4140 FOR LOC=LOOP TO LOOP+4 STEP 2
4150 GOSUB 4470
4160 M=MX
4170 N=NX
4180 V1=256*M+N
4190 GOSUB 3580
4200 IF LOC=LOOP+4 THEN 4230
4210 IF LOC>=B-1 THEN 4260
4220 PRINT #F:">";HEX$;",";
4230 NEXT LOC
4240 PRINT #F:">";HEX$
4250 NEXT LOOP
4260 PRINT #F:">";HEX$
4270 GOTO 420
4280 REM  DISPLAY TEXT ON SCREEN
4290 FOR LOOP=A TO B STEP 14
4300 V1=LOOP
4310 GOSUB 3580
4320 PRINT #F:HEX$;" TEXT '";
4330 FOR LOC=LOOP TO LOOP+13
4340 GOSUB 4470
4350 M=MX
4360 IF (M<127)+(M>31)=-2 THEN 4380
4370 M=63
4380 PRINT #F:CHR$(M);
4390 IF LOC=B THEN 4430
4400 NEXT LOC
4410 PRINT #F:"'"
4420 NEXT LOOP
4430 PRINT #F:"'"
4440 GOTO 420
4450 REM
4460 REM   PEEK ROUTINE
4470 IF LOC<32768 THEN 4500
4480 LOCX=LOC-65536
4490 GOTO 4510
4500 LOCX=LOC
4510 CALL PEEK(LOCX,MX,NX)
4520 RETURN
4530 END
```

Just Assemble Melody:

Music in Mini Memory

Using the Machine Language Routine

This program was intended to facilitate the construction of sound lists for Assembly Language applications, but it may also be used in conjunction with TI BASIC programs if some care is exercised. Because the sound list will play in its entirety, even while the BASIC interpreter continues, your program can go on to other things while the music plays. However, because TI BASIC has no idea what is going on, there may be unpredictable side effects, especially when BASIC is manipulating strings or displaying graphics on the screen. [See the Editor/Assembler manual, page 312, for more information.—Ed.] Any CALL SOUND statement will stop the execution of a sound list. Here is a sample program which will play a sound list over and over until a key is pressed, then interrupt it (much like the techniques used in TI's *Tombstone City*):

```
100 CALL LINK("PLAY")
110 CALL PEEK(-31794,N)        Is it done?
120 IF N=0 THEN 100            Yes, play it again.
130 CALL KEY(0,K,S)            Key pressed?
140 IF S=0 THEN 110            No, check sound again.
150 CALL SOUND(-1,-1,30)       Stop the sound list.
 .
 . program continues
 .
```

The PEEK statement (line 110) reads a byte at >83CE. This will be zero when the interrupt routine has finished the sound list.

You can save and reload the PLAY subprogram and any sound list you generate with EASYBUG. Be sure to save everything from >7000 to >7FFF. This ensures that you have the entry point to the PLAY subprogram, and can play the sound list using the short program segment above.

Here, in the form used with the *Line-by-Line Assembler*, is an Assembly Language listing of the "PLAY" subprogram that the *Music Assembler* loads:

```
ST   EQU    >837C
     AORG   >7FBE
TA   DATA   0              Any value poked into this address
                           will be used as the address of the
                           sound list.
H1   BYTE   >01
     EVEN

PL   MOV    @TA,R1         Get table length pointer.
     MOV    R1+,R2         R1 is now start of sound list.
     MOV    R2,@>830C      Copy length to GPL parameter
                           address.

     BLWP   @>6018         Get string space routine.
     DATA   >0038          Get address of allocated space.
     MOV    @>831C,R0      Move sound list to VDP RAM.
     BLWP   @>6028

     LIMI   0              Play list
     MOV    R0,@>83CC
     SOCB   @H1,@>83FD
     MOVB   @H1,@>83CE
     LIMI   2
     CLR    @ST            Return to BASIC.
     RT
     TEXT   'PLAY
     DATA   PL
     AORG   >701C
     DATA   >7FF8, >7FF8
     END
```

It would be possible to put the program title in the REF/DEF table in this way because the program begins at >7FBE and uses up all but the last eight bytes of memory. The last two lines set *both* the First *and* Last Free Addresses in Medium Memory to >7FF8. The Last Free Address indicates the start of the REF/DEF table.

TI-99/4A

```
100 REM * * * * * * * * * * * * * * * * * *
110 REM *  MUSIC ASSEMBLER  *
120 REM * * * * * * * * * * * * * * * * * *
130 REM  BY CLEON CHAPEN
140 REM  HOME COMPUTER MAGAZINE
150 REM  VERSION 4.1.1
160 REM  TI BASIC, TI MINI MEMORY
170 REM *  MINI MEMORY
180 REM *  FUNCTIONS
190 REM *
200 DEF H(N)=INT(N/16)
210 DEF L(N)=N-H(N)*16
220 DEF CODE(N)=INT(111860.8/N+.5)
230 REM *
240 REM *  ARRAYS
250 REM *
260 DIM SD(8),NOS(68),FRE(68)
270 REM *
280 REM *  DATA BLOCK
290 REM *
300 REM *  PRE-DEFINED NOTES
310 REM *
320 DATA A#0,116.54,A0,110
330 DATA B0,123.47,C#1,138.59
340 DATA C1,130.81,D#1,155.56
350 DATA D1,146.83,E1,164.81
360 DATA F#1,185,F1,174.61
370 DATA G#1,207.65,G1,196
380 DATA A#1,233.08,A1,220
390 DATA B1,246.94,C#2,277.18
400 DATA C2,261.63,D#2,311.13
410 DATA D2,293.66,E2,329.63
420 DATA F#2,369.99,F2,349.23
430 DATA G#2,415.30,G2,392
440 DATA A#2,466.16,A2,440
450 DATA B2,493.88,C#3,554.37
460 DATA C3,523.25,D#3,622.25
470 DATA D3,587.33,E3,659.26
480 DATA F#3,739.99,F3,698.46
490 DATA G#3,830.61,G3,783.99
500 DATA A#3,932.33,A3,880
510 DATA B3,987.77,C#4,1108.73
520 DATA C4,1046.5,D#4,1244.51
530 DATA D4,1174.66,E4,1318.51
540 DATA F#4,1479.98,F4,1396.91
550 DATA G#4,1661.22,G4,1567.98
560 DATA A#4,1864.66,A4,1760
570 DATA B4,1975.53,C#5,2217.46
```

TI-99/4A

```
580 DATA C5,2093,D#5,2489.02
590 DATA D5,2349.32,E5,2637.02
600 DATA F#5,2959.96,F5,2793.83
610 DATA G#5,3322.44,G5,3135.96
620 DATA A#5,3729.31,A5,3520
630 DATA B5,3951.07,C#6,4434.92
640 DATA C6,4186.01,D#6,4978.03
650 DATA D6,4698.64,E6,5274.04,F6,5587.
65
660 REM *
670 REM *  MACHINE ROUTINE
680 REM *
690 DATA 1,0,192,96,127,190,192,177
700 DATA 200,2,131,12,4,32,96,24,0,56,1
92,32,131,28,4,32,96,40,3,0,0,0,200
710 DATA 0,131,204,248,32,127,192,131,2
53,216,32,127,192,131,206
720 DATA 3,0,0,2,4,224,131,124,4,91
730 DATA 80,76,65,89,32,32,127,194
740 REM *
750 REM *  SETUP
760 REM *
770 CALL CLEAR
780 PRINT TAB(7);"MUSIC ASSEMBLER":::::
::::
790 FOR I=0 TO 68
800 READ NOS(I),FRE(I)
810 NEXT I
820 GOSUB 2890
830 CALL CLEAR
840 D$="."
850 H$="0123456789ACBDEF"
860 CALL PEEK(28700,H1,L1)
870 FFM=H1*256+L1
880 LC=FFM+2
890 CALL CLEAR
900 REM *
910 REM *  MAIN LOOP
920 REM *
930 GOSUB 2400
940 IF FL=99 THEN 1870
950 OPTR=0
960 FOR I=0 TO 8
970 SD(I)=0
980 NEXT I
990 FOR I=0 TO 8
1000 NPTR=POS(I$,D$,OPTR+1)
1010 IF NPTR=0 THEN 1050
```

```
1020 IF OPTR+1=NPTR THEN 1290
1030 S$=SEG$(I$,OPTR+1,NPTR-OPTR-1)
1040 GOTO 1070
1050 S$=SEG$(I$,OPTR+1,5)
1060 IF (S$=" ")+(S$="     ")THEN 1370
1070 IF I=7 THEN 1130
1080 IF (I=1)+(I=3)+(I=5)THEN 1090 ELSE
     1190
1090 IF (ASC(S$)>64)*(ASC(S$)<72)THEN 11
     00 ELSE 1120
1100 GOSUB 2290
1110 GOTO 1300
1120 IF (ASC(S$)>47)*(ASC(S$)<58)THEN 11
     90
1130 IF SEG$(S$,1,1)<>"-" THEN 1240
1140 FOR J=2 TO LEN(S$)
1150 IF (ASC(SEG$(S$,J,1))>47)*(ASC(SEG$
     (S$,J,1))<58)THEN 1170
1160 GOTO 1240
1170 NEXT J
1180 IF VAL(S$)>-9 THEN 1260 ELSE 1240
1190 FOR J=1 TO LEN(S$)
1200 IF (ASC(SEG$(S$,J,1))>47)*(ASC(SEG$
     (S$,J,1))<58)THEN 1220
1210 GOTO 1240
1220 NEXT J
1230 GOTO 1260
1240 PRINT :"* BAD VALUE: ";S$:"   TRY AG
     AIN...":
1250 GOTO 930
1260 SD(I)=VAL(S$)
1270 GOSUB 2680
1280 GOTO 1300
1290 SD(I)=0
1300 OPTR=NPTR
1310 NEXT I
1320 REM *
1330 REM * MAKE SOUND LIST
1340 REM *
1350 REM * DURATION
1360 REM *
1370 DU=INT(SD(0)*255/4250)
1380 DU$=CHR$(DU-(DU=0))
1390 REM *
1400 REM * SOUND GENERATORS
1410 REM *
1420 G1$=""
1430 G2$=""
1440 G3$=""
1450 NC$=""
1460 NA$=""
1470 IF SD(1)=0 THEN 1510
1480 V=CODE(SD(1))
1490 G1$=CHR$(128+L(V))&CHR$(H(V))
1500 G1$=G1$&CHR$(144+INT(SD(2)/2))
1510 IF SD(3)=0 THEN 1550
1520 V=CODE(SD(3))
1530 G2$=CHR$(160+L(V))&CHR$(H(V))
1540 G2$=G2$&CHR$(176+INT(SD(4)/2))
1550 IF SD(5)=0 THEN 1600
1560 V=CODE(SD(5))
1570 G3$=CHR$(192+L(V))&CHR$(H(V))
1580 G3$=G3$&CHR$(208+INT(SD(6)/2))
1590 REM *
1600 REM * NOISE CONTROL
1610 REM *
1620 IF SD(7)=0 THEN 1730
1630 IF SD(7)<-4 THEN 1660
1640 TY=0
1650 GOTO 1680
1660 TY=4
1670 SD(7)=SD(7)+4
1680 NC$=CHR$(224+TY+ABS(SD(7))-1)
1690 NA$=CHR$(240+INT(SD(8)/2))
1700 REM *
1710 REM * FINISH AND LOAD
1720 REM *
1730 SOUND$=G1$&G2$&G3$&NC$&NA$&DU$
1740 L$=CHR$(LEN(SOUND$)-1)
1750 SOUND$=L$&SOUND$
1760 LF=LF+LEN(SOUND$)
1770 SAVELC=LC
1780 FOR I=1 TO LEN(SOUND$)
1790 CALL LOAD(LC,ASC(SEG$(SOUND$,I,1)))
1800 LC=LC+1
1810 IF LC>32701 THEN 2850
1820 NEXT I
1830 GOTO 930
1840 REM *
1850 REM * RESET POINTERS
1860 REM *
1870 CALL LOAD(LC,4,159,191,223,255,0)
1880 LF=LF+6
1890 LC=LC+6
1900 IF LC/2=INT(LC/2)THEN 1920
1910 LC=LC+1
1920 FH=INT(LF/256)
1930 FL=LF-FH*256
1940 CALL LOAD(FFM,FH,FL)
1950 LCH=INT(LC/256)
1960 LCL=LC-LCH*256
1970 CALL LOAD(28700,LCH,LCL)
1980 REM *
1990 REM * PLAY LIST?
2000 REM *
2010 CALL CLEAR
2020 PRINT "* ASSEMBLY COMPLETED":
2030 INPUT "* PLAY LIST? (Y/N) ":I$
2040 IF ASC(I$)=89 THEN 2070
2050 IF ASC(I$)=78 THEN 2210
2060 GOTO 2010
2070 RESTORE 690
2080 LOC=32704
2090 CALL LOAD(LOC-2,H1,L1)
2100 PRINT "* LOADING MACHINE ROUTINE"
2110 FOR I=0 TO 63
2120 READ V
2130 CALL LOAD(LOC+I,V)
2140 NEXT I
2150 PRINT "* PLAYING"
2160 CALL LOAD(28702,127,248)
2170 CALL LINK("PLAY")
2180 PRINT :"* PRESS ANY KEY TO PROCEED"
2190 CALL KEY(0,K,S)
2200 IF S=0 THEN 2190
2210 PRINT :"* COMPLETED SOUND TABLE
         LOCATED AT ADDRESS";FFM
2220 GOSUB 2570
2230 STOP
2240 REM *
2250 REM * SUBROUTINES
2260 REM *
2270 REM * SEARCH FOR NOTE
2280 REM *
2290 FOR J=0 TO 68
2300 IF NO$(J)=S$ THEN 2350
2310 NEXT J
2320 PRINT :"* UNKNOWN NOTE NAME: ";S$::
2330 INPUT "    RETYPE NOTE: ":S$
2340 GOTO 2290
2350 SD(I)=FRE(J)
2360 RETURN
2370 REM *
2380 REM * KEYBOARD INPUT
2390 REM *
2400 IF OP THEN 2430
2410 PRINT "* ENTER SOUND LISTS"::"* ENTE
     R NULL STRING TO END"::
2420 OP=-1
2430 INPUT I$
2440 IF I$="" THEN 2450 ELSE 2470
2450 FL=99
2460 RETURN
2470 IF I$="REDO" THEN 2480 ELSE 2500
2480 LC=SAVELC
2490 GOTO 2430
2500 IF SEG$(I$,LEN(I$),1)="." THEN 2530
2510 PRINT :"* FINAL PERIOD MISSING.
         TRY AGAIN...":
2520 GOTO 2400
2530 RETURN
2540 REM *
2550 REM * MAKE HEX ADDRESS
2560 REM *
2570 A$=""
2580 FOR D=12 TO 0 STEP -4
2590 N=INT(FFM/(2^D))
2600 FFM=FFM-N*(2^D)
2610 A$=A$&SEG$(H$,N+1,1)
2620 NEXT D
2630 PRINT "   (HEX >";A$;")"
2640 RETURN
2650 REM *
2660 REM * CHECK RANGES
2670 REM *
2680 ON I+1 GOTO 2690,2720,2750,2720,275
     0,2720,2750,2780,2750
2690 MIN=1
2700 MAX=4250
2710 GOTO 2800
2720 MIN=110
2730 MAX=44733
2740 GOTO 2800
2750 MIN=0
2760 MAX=30
2770 GOTO 2800
2780 MIN=-8
2790 MAX=-1
2800 IF (SD(I)<=MAX)*(SD(I)>=MIN)THEN 28
     10 ELSE 2820
2810 RETURN
2820 PRINT :"* BAD VALUE: ";SD(I):"   ";S
     TR$(MIN);" TO ";STR$(MAX):"   IS THE
     VALID RANGE"::
2830 INPUT "   RETYPE BAD NUMBER: ":SD(I)
2840 GOTO 2800
2850 REM *
2860 REM * OUT OF MEMORY
2870 REM *
2880 PRINT ::::"**** MEMORY FULL ****"::
     ::
2890 REM *
2890 END
2900 REM *
2910 REM * CHECK MACHINE
2920 REM * DATA - DELETE
2930 REM * WHEN DATA IS
2940 REM * VERIFIED
2950 REM *
2960 FOR I=0 TO 63
2970 READ V
2980 CHECK=CHECK+V
2990 NEXT I
3000 IF CHECK=5790 THEN 3010 ELSE 3020
3010 RETURN
3020 PRINT ::"*** ERROR IN MACHINE ROUTI
     NE"::
```

FRANCOMPUTER

CLUB
TEXAS INSTRUMENTS
TI 99/4A

PROGRAMMI - SCAMBI - INSEGNAMENTO

CORSO FOGAZZARO 174
(0444) VICENZA 42678
36100